School of Computing and Information Systems,
The University of Melbourne

Autoregressive Generative Models
and
Multi-Task Learning
with
Convolutional Neural Networks

Florin Schimbinschi

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

October 2018

Abstract

At a high level, sequence modelling problems are of the form where the model aims to predict the next element of a sequence based on neighbouring items. Common types of applications include time-series forecasting, language modelling, machine translation and more recently, adversarial learning. One main characteristic of such models is that they assume that there is an underlying learnable structure behind the data generation process, such as it is for language. Therefore, the models used have to go beyond traditional linear or discrete hidden state models.

Convolutional Neural Networks (CNNs) are the de facto state of the art in computer vision. Conversely, for sequence modelling and multi-task learning (MTL) problems, the most common choice are Recurrent Neural Networks (RNNs). In this thesis I show that causal CNNs can be successfully and efficiently used for a broad range of sequence modelling and multi-task learning problems. This is supported by applying CNNs to two very different domains, which highlight their flexibility and performance: 1) traffic forecasting in the context of highly dynamic road conditions - with non-stationary data and normal granularity (sampling rate) and a high spatial volume of related tasks; 2) learning musical instrument synthesisers - with stationary data and a very high granularity (high sampling rate - raw waveforms) and thus a high temporal volume, and conditional side information.

In the first case, the challenge is to leverage the complex interactions between tasks while keeping the streaming (online) forecasting process tractable and robust to faults and changes (adding or removing tasks). In the second case, the problem is highly related to language modelling, although much more difficult since, unlike words, multiple musical notes can be played at the same

time, therefore making the task much more challenging.

With the ascent of the Internet of Things (IoT) and Big Data becoming more common, new challenges arise. The four V's of Big Data (Volume, Velocity, Variety and Veracity) are studied in the context of multi-task learning for spatio-temporal (ST) prediction problems. These aspects are studied in the first part of this thesis. Traditionally such problems are addressed with static, non-modular linear models that do not leverage Big Data. I discuss what the four V's imply for multi-task ST problems and finally show how CNNs can be set up as efficient classifiers for such problems, if the quantization is properly set up for non-stationary data.

While the first part is predominantly data-centric, focused on aspects such as Volume (is it useful?) and Veracity (how to deal with missing data?) the second part of the thesis addresses the Velocity and Variety challenges. I also show that even for prediction problems set up as regression, causal CNNs are still the best performing model as compared to state of the art algorithms such as SVRS and more traditional methods such as ARIMA. I introduce TRU-VAR (Topologically Regularized Universal Vector AutoRegression) which, as I show, is a robust, versatile real-time multi-task forecasting framework which leverages domain-specific knowledge (task topology), the Variety (task diversity) and Velocity (online training).

Finally, the last part of this thesis is focused on generative CNN models. The main contribution is the SynthNet architecture which is the first capable of learning musical instrument synthesisers end-to-end. The architecture is derived by following a parsimonious approach (reducing complexity) and via an in-depth analysis of the learned representations of the baseline architectures. I show that the 2D projection of each layer gram activations can correspond to resonating frequencies (which gives each musical instrument it's timbre). SynthNet trains much faster and it's generation accuracy is much higher than the baselines. The generated waveforms are almost identical to the ground truth. This has implications in other domains where the the goal is to generate data with similar properties as the data generation process (i.e. adversarial examples).

In summary, this thesis makes contributions towards multi-task spatio-temporal

time series problems with causal CNNs (set up as both classification and regression) and generative CNN models. The achievements of this thesis are supported by publications which contain an extensive set of experiments and theoretical foundations.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the degree of Doctor of Philosophy except where indicated in the Preface,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is fewer than 80,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Florin Schimbinschi

Preface

This thesis has been written at the School of Computing and Information Systems, The University of Melbourne. The major parts of the thesis are Chapters 3, 4 and 5. These are based on published proceeding or submitted papers and I declare that I am the primary author and have contributed $> 50\%$ in all of the following:

1. F. Schimbinschi, V.X. Nguyen, J. Bailey, C. Leckie, H. Vu, R. Kotagiri, "Traffic forecasting in complex urban networks: Leveraging big data and machine learning" in *International conference on Big Data*, (IEEE BigData) pp. 1019-1024, IEEE, 2015.

2. F. Schimbinschi, L. Moreira-Matias, V.X. Nguyen, J. Bailey, "Topology-regularized universal vector autoregression for traffic forecasting in large urban areas" in *Expert Systems with Applications* (ESWA) vol. 82, pp. 301-316, Pergamon, 2017.

3. F. Schimbinschi, C. Walder, S.M. Erfani, J. Bailey, "SynthNet: Learning synthesisers end-to-end." under review *International Conference on Learning Representations*, (ICLR) 2019.

Acknowledgements

First and foremost I would like to thank my family and friends for their support. A very special thanks to Elena - without her encouragement and optimism my PhD experience would have been very different.

I would also like to thank my supervisors, Prof. James Bailey, Dr. Sarah Monazam Erfani and Dr. Xuan Vinh Nguyen for their wisdom and for the experience that I have gained while working as a research student at the University of Melbourne. I would also like to thank all my colleagues and the people that I've met in the department throughout this journey.

As a PhD candidate I have also collaborated with people from outside the university, namely Luis Moreira-Matias from NEC Labs Europe and Christian Walder from CSIRO-Data61 Canberra, to whom I am thankful for their research perspective and expertise.

Finally, my research was made possible by CSIRO-Data61 (formerly NICTA). I am thankful for the computational resources that I extensively used, especially in the last part of my candidature.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The massive amount of data that is being collected from today's devices is tremendous. This fuels the need for ever more efficient and easily deployable algorithms. Sequence modelling and multi-task learning (MTL) problems are ubiquitous. In the current information age, connectivity is a core design component of any physical device, enabling the control and exchange of data. Currently, there are many networks of physical devices such as home appliances, cars, wind farms, and in general virtually any sort of device that has embedded networking electronic components. The network of such connected devices has come to be known as the Internet of Things (IoT). Consequently, there is an abundance of time series data that can be collected from such networks (Big Data).

Traditionally, recurrent neural networks (RNNs) have been the go-to generative method for sequential data. Although CNNs are practically ubiquitous in computer vision problems, their full potential has not yet been explored for generative models and multi-task learning. In this thesis I show that CNNs can be not only highly efficient and accurate but also versatile to deploy and can be applied to a broad range of problems, from detection, forecasting to generative models and also to generating adversarial examples. In subsection 2.1.3 I show that the autoregressive deployment of linear models, Feed Forward Neural Networks (FFNNs), as well as kernel methods, can also be interpreted as causal convolutional models, exemplified specifically for CNNs. (subsection 2.1.3 also

provides the notation and all the neural network architectural components used throughout this thesis.) Then these are used in a sparsely grouped MTL framework applied to forecasting. in Chapter 3 and Chapter 4. Finally, CNNs, MTL and generative models come together in Chapter 5 where SynthNet is introduced. This model makes use of several CNN architectural blocks and is trained in a MTL framework with an auxiliary task which improves training.

MTL is essential for applications such as diagnostic (detecting anomalies such as faulty devices), prediction (power output of a wind turbine) and control (best route to take while driving). And yet these are only some of the domains where these are central. Any task that involves prediction or control with inter-related time series data can be modelled as a MTL problem. MTL also has applications in optimal control: modeling the kinematics of a robotic arm can be formulated as a multi-task learning problem since the joints (motors) are connected. The latter can also be seen as a sequence to sequence (seq2seq) application such as language translation. Other examples include fault prediction in computer networks(i.e. anomaly detection) or fraud protection for electronic payments.

Generative models open up many possibilities for IoT. There are many applications including speech synthesis, text analysis and synthesis, semi-supervised learning and model-based control and some are yet to be discovered. Most of these come at a high resolution. The most obvious ones are in natural language processing, machine translation (and as exemplified before in control problems), text to speech, learning synthesizers, voice cloning, generating audio or video based on other data (e.g. translate video speech directly to foreign language based on video lip reading). To connect generative models with IoT and MTL, these can be used to generate artificial or fake data (also known as adversarial examples). This is useful in circumstances when one would like to defend against possible attackers by creating fake sensors or adding fake data in order to confuse attackers. This is useful since, for example DoS attacks are targeted at peak load times.

Many large modern cities have placed sensors under the roads at intersections, in order to record the number of cars that pass through, at one point in time. Then, this data can be used, for example, to control the traffic lights (over

the entire network) and in effect prevent or alleviate congestion. Since there are multiple prediction points that are interdependent, this provides an opportunity to study challenging MTL problems in general. The VicRoads dataset (Figure 1.1) was recorded in Melbourne over 5 years. It spans a very large and dense area, is diverse and challenging to tackle since not all sensors were installed at the same time. One other characteristic of this dataset is that approximately half of the sensor stations (tasks) can have more than 50% data missing, which makes the prediction problem more difficult. This dataset is used in Chapter 3 and Chapter 4 to benchmark causal CNNs against other methods such as SVMs, Boosting, Trees, Naive Bayes, K-nearest Neighbours, Linear/Logistic regression and ARIMA.

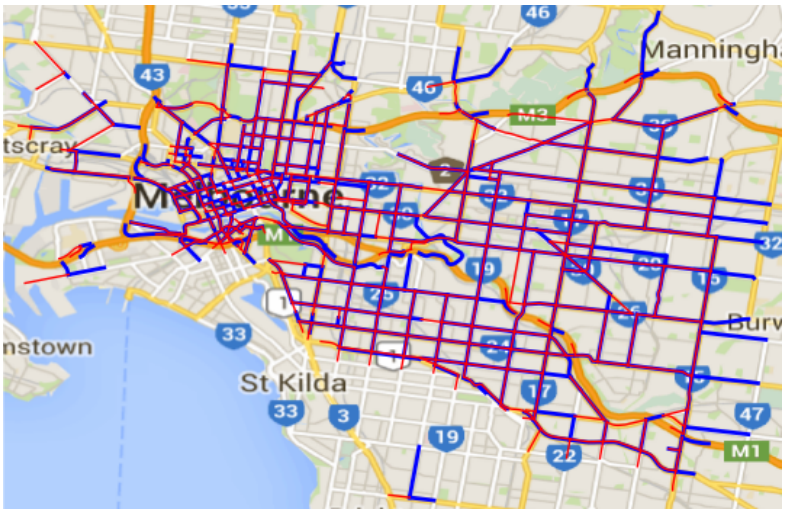


Figure 1.1: VicRoads dataset - 1000+ sensors over 6 years of data over a very broad and dense area.

Although this dataset has a very large number of tasks (1000+) the resolution (granularity) is not very high. Furthermore, it is not clear how to interpret the data generation process. On the other hand, music comes at a high resolution (standard commercial quality is at 44.1 thousand frames a second) and music theory and the physics behind the timbre of musical instruments is well understood.

For example, when the note A (110Hz) string is plucked on a guitar, the string vibrates back and forth at many different frequencies at the same. The

lowest is 110 Hz - this is the fundamental frequency (for a timbreless instrument this would be a pure sine wave). Since the string is fixed at both ends, it can only vibrate in multiples of the fundamental frequency: halves are the second harmonic at 220 Hz, thirds are the third harmonic at 330 Hz and so on. This is called the Harmonic series (or the Overtone series) and it is at the basis of how music is structured. What gives an instrument it's specific timbre, is due to the physics of the instrument (e.g. guitar, trumpet, etc), the materials used and the slight imperfections in the build which are even more subtle. It should also be understood that the overtone series are not necessarily an even multiple of the fundamental frequency.

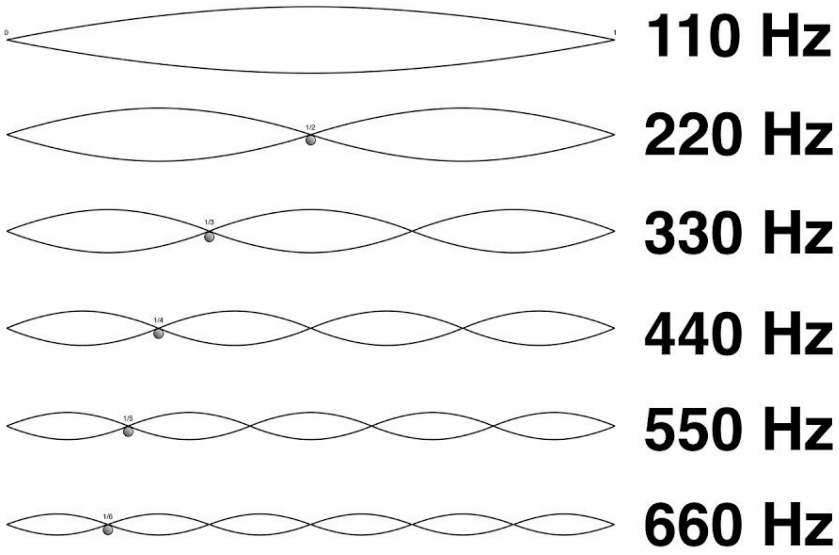


Figure 1.2: Timbre is given by the overtone series.

However, regardless of the instrument, the fundamental frequency is the same, since that is the musical note that we hear. This brings up the opportunity to study MTL and generative models for high resolution time series problems since the content (sequence of notes - fundamental frequencies) can be understood separately from the timbre (overtone series - specific for each musical instrument). Chapter 5 uses synthesized (registered music data) in order to study the learned representations of convolutional autoregressive generative models, trained with an auxiliary MTL task.

Thesis structure Chapter 2 builds a foundation for better understanding of the proceeding chapters, provides the common notation and a literature review on the existing related work. Chapter 2 is central to all the work in this thesis and especially subsection 2.1.3 which discusses deep causal convolutional neural networks (CNNs). In Chapter 2 I also discuss parallels among autoregressive models and causal CNNs, autoregressive generative models, MTL regularization and optimization methods. I also recommend reading subsection 2.1.4 prior to Chapter 5. Chapter 3 approaches the four V's of Big Data in the context of multi-task learning with spatio-temporal (ST) prediction problems modelled as classification (peak traffic forecasting). Several methods are benchmarked against causal CNNs.

Chapter 4 builds on top of this work and extends it to continuous outputs (i.e. vector autoregression regression - VAR). In addition, Chapter 4 also discusses scalability and modularity for ST MTL problems where the structure of the graph describing the task relationship is known. Related is section 2.2 in Chapter 2 which provides a detailed theoretical discussion on regularization for MTL and shows that VAR is a special case of MTL.

While Chapters 3 and 4 are dedicated to MTL, Chapter 5 is focused on language models for music and introduces SynthNet which a WaveNet derived autoregressive generative CNN. Music is a good domain to study the representations learned by autoregressive CNNs since the structure of instrument harmonics is well known and hence allows the methodical study of learned representations.

1.1 Research contributions

In this thesis I show that CNNs can be very effective and versatile when applied to MTL and generative models. The main advantage is that they are relatively easier to train (vs RNNs) and are faster to train. Despite some architectures having a very large receptive field compared to any other models, they have a very low computational overhead. The large receptive field is critical for high resolution (high sampling rate) signals such as raw audio.

The contributions of Chapter 3 are approached from a data centric perspec-

tive of the four V's of Big Data (Volume, Variety, Velocity, Veracity). I show that indeed Volume can have a beneficial impact towards performance in the context of MTL for ST problems. I ask the questions whether recent data is more useful on its own and whether Big Data helps even if old data is used. The latter turns out to be true. However, Volume for ST problems specifically, can be the main challenge since the spatial Volume represents the number of tasks. Even with a set of over 1000 such tasks (each with 4 years of data),

I demonstrate in Chapter 3 that causal CNNs outperform other state of the art methods. Velocity is directly linked with the problem setup and modeling when predictions are to be done simultaneously for all tasks (the number of tasks can also change in time). I also show in Chapter 3 that modularity is key to the latter and that causal CNNs can be trained efficiently online (in real-time - Velocity) thus are the best choice in the context of spatio-temporal MTL forecasting with Big Data. Furthermore, unlike RNNs, these are easy to deploy or redistribute (can be 'cloned' easily). The Variety dimension is directly related to task diversity and I show that exchanging data between tasks is indeed beneficial. As to the Veracity component, Chapter 4 shows that augmenting missing data with contextual average trends only marginally increases accuracy.

Furthermore, Chapter 4 shows that for this particular MTL problem, detrending the data is not beneficial since information that the causal CNNs leverage is removed. Chapter 4 shows that the MTL traffic prediction problem is equivalent to simultaneously solving several sparsely grouped MTL problems. The theoretical framework is benchmarked with state of the art (SOA) models. The results show that causal CNNs outperform other linear and nonlinear models. This is arguably controversial since traditional linear models usually benefit from such methods. This shows that causal CNNs in the context of MTL for ST problems are more versatile and robust towards missing data.

The main contribution of Chapter 4 is the TRU-VAR framework which generalizes MTL for ST problems and shows that solving grouped MTL problems can be equivalent and more efficient to solving a single MTL problem. This chapter also shows that using the topological structure of roads in MTL problems is a better approach towards introducing sparse groups since correlation based methods are not accurate with physical location. Several methods are

benchmarked within this framework on two different datasets, and for both, the causal CNNs yield the best results.

In conclusion, Chapter 4 shows that TRU-VAR is able to scale well to large datasets, is robust and furthermore is easily deployable with new sensor installations. The adjacency matrix used for generating sparse MTL task groups should be chosen carefully and should be done as a function of the dataset and domain. Furthermore, high resolution data (temporal as well as spatial) is essential. Missing data can be an issue and augmenting it does not significantly increase performance, however it should be explicitly marked in order to distinguish it from real events (e.g. congestion in traffic).

The last part of this thesis – Chapter 5 is dedicated to autoregressive generative CNNs. It introduces the SynthNet algorithm which is able to generate high fidelity audio based on only 9 minutes of training data per instrument timbre. While the original algorithm (WaveNet) from which this work was derived was able to generate speech that sounded realistic, the waveforms were very different to the ground truth. SynthNet is able to generate audio so accurately that the generated waveforms are almost identical to the ground truth. This is very significant since music is more complex than spoken word.

In addition SynthNet trains and converges faster than the baselines since it has fewer parameters. Chapter 5 also gives an explanation of the structure of the learned representations of autoregressive CNNs, which advances the understanding of CNNs for audio closer to the ones in images. This was done by testing the hypothesis that the first causal layer learns fundamental frequencies. This was validated empirically, arriving at the SynthNet architecture.

The method is able to simultaneously learn the characteristic harmonics of a musical instrument (timbre) and a joint embedding between notes and the corresponding fundamental frequencies. This has implications in many fields since the ability to generate signals that match the ground truth with a high fidelity is a very desirable property. Previous methods were not able to do so and were focused purely on the qualitative (how does the audio sound like) measurements while the actual generated data differed from the ground truth.

1.1.1 Outline of contributions

The contributions of Chapter 3 mainly stem from the experiments:

(i) modeling prediction as a multi-task learning problem is beneficial;

(ii) the spatio-temporal representation is one of the central issues;

(iii) predicting only on weekdays is easier and separate predictors can be deployed separately for weekends or each day of the week;

(iv) adjusting the receptive field size and proximity lowers error;

(v) for classification problems, the quantization of real-valued data is a central issue and should either be avoided or thresholding should be set dynamically.

The contributions of Chapter 4 are as follows:

(i) I propose learning Topology-Regularized Universal Vector Autoregression (TRU-VAR), a novel framework that is based on the spatio-temporal dependences between multiple sensor stations;

(ii) The extension of TRU-VAR to CNNs and other nonlinear universal function approximators over the existing state of the art machine learning algorithms, resulting in an exhaustive comparison;

(iii) The evaluations performed on two large scale real world datasets, which are different: one is sparse high granularity, the other is dense and has slightly lower granularity;

(iv) Comprehensive coverage of the literature, and an exploratory analysis considering data quality, preprocessing and possible heuristics for choosing the topology-designed adjacency matrix (TDAM).

Finally, Chapter 5 makes the following contributions:

(i) I show that musical instrument synthesizers can be learned end-to-end based on raw audio and a binary note representation, with minimal training data;

(ii) Multiple instruments can be learned by a single model;

(iii) I give insights into the representations learned by dilated causal convolutional blocks;

(iv) I propose SynthNet, which provides substantial improvements in quality and training time and convergence rate compared to previous work;

(v) I demonstrate (Figure 5.8) that the generated audio is practically identical to the ground truth;

(vi) The benchmarks against existing architectures contains an extensive set of experiments spanning over three sets of hyperparameters, where I control for receptive field size;

(vii) I show that the RMSE of the Constant-Q Transform (RMSE-CQT) is highly correlated with the subjective listening mean opinion score (MOS);

(viii) I find that reducing quantization error via dithering is a critical preprocessing step towards generating the correct melody and learning the correct pitch to fundamental frequency mapping.

Chapter 2

Background

This chapter provides an introduction and an overview of the core topics used throughout the rest of the chapters. In section 2.1 I give an introduction to simple linear models, then I extend the scope to kernel methods and finally discuss Feed Forward Neural Networks (FFNNs). I show that autoregressive models can also be interpreted as causal convolutional models, as exemplified for converting FFNNs to Causal Convolutional Neural Networks (CNNs). subsection 2.1.4 discusses the WaveNet architecture in detail based on the previous sections. Then, I give an overview of MTL regularization in section 2.2 showing that regularization can be used to share parameters between related tasks and that tasks can be grouped based on a known prior structure. In section 2.3 I discuss gradient based optimization methods for fitting these models. Finally, section 2.4 provides an extensive literature review on autoregressive models, MTL and generative models for time series.

2.1 Autoregressive and sequence models

It is common to model a time series as an autoregressive process. For a time series $\boldsymbol{x}$, each observation x_t can be conditioned on the observations at all previous time steps $\boldsymbol{x}_{<t} = \{x_{t-1}, x_{t-2}, \ldots, x_1\}$. The joint probability of a time series $\boldsymbol{x} = \{x_1, \ldots, x_T\}$ is then factorized as follows:

$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t \mid \boldsymbol{x}_{<t}) \tag{2.1}$$

Traditionally, the Autoregressive Moving Average (ARMA) model is used for modelling time series. This stochastic linear model is composed of an AutoRegressive (AR) and a Moving Average (MA) component. An AR model assumes the predicted value to be a linear combination of Δ_p past observations:

$$y_t = \theta_0 + \sum_{i=1}^{\Delta_p} \theta_i x_{t-i} + \epsilon_t \quad \text{where } \epsilon \in \mathcal{N}(0, \sigma) \tag{2.2}$$

where θ_0 is the intercept or bias, y_t is the scalar response value, x_i are the past time series observations (or explanatory variables) and ϵ_t is the random error.

For AR models, Δ is known as the order of the model, and is also known as the *lag*. In the neural network literature Δ is referred to as the size of the *receptive field*.

While an AR(Δ_p) model regresses against past values of the time series, a MA(Δ_q) model uses the past errors as the explanatory variables (where μ is the mean of the time series):

$$y_t = \mu + \sum_{j=1}^{\Delta_q} \phi_j \epsilon_{t-j} + \epsilon_t \tag{2.3}$$

These can be finally combined into the ARMA model:

$$y_t = \theta_0 + \epsilon_t + \sum_{i=1}^{\Delta_p} \theta_i x_{t-i} + \sum_{j=1}^{\Delta_q} \phi_j \epsilon_{t-j} \tag{2.4}$$

ARMA models, are applied in Chapter 4 and no longer discussed here, for a detailed discussion see [19].

Linear regression For convenience I denote the scalar response value $y = x_t$ and write the vector of explanatory variables as $\boldsymbol{x}_\Delta = (1, x_{t-1}, x_{t-2}, \ldots, x_{t-\Delta})$ and the parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_\Delta)$, also including the intercept. Then, the AR model can be rewritten as classic linear regression:

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}_\Delta \tag{2.5}$$

where the total number of observations reduces to $N = T - \Delta$.

To lighten the notation and generalize, any autoregressive model f parametrized by $\boldsymbol{\theta}$ will be written from now on as $y = f(\boldsymbol{x}, \boldsymbol{\theta})$, as exemplified in Equation 2.5, which is identical to the AR model described in Equation 2.2. The conversion from an AR process to classic regression is depicted in Figure 2.1 where the grey box represents Δ. The AR process is also described as a causal convolution in Figure 2.4.

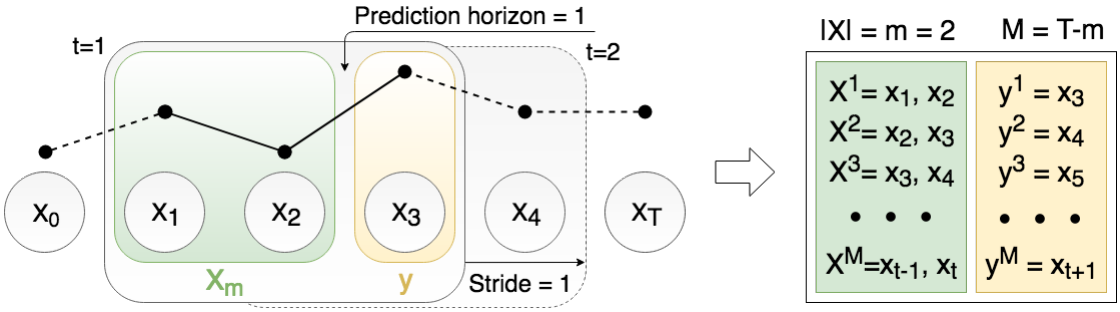


Figure 2.1: Converting AR to classic regression via a sliding window.

Logistic regression For regression the goal is to solve problems of the form $f : X \to Y \subseteq \mathbb{R}$ while for multiclass classification the goal is to find $f : X \to Y = \{1, 2, \ldots, C\}$ where C is the number of classes. For classification, the softmax function ψ extends linear to logistic regression by representing a probability distribution over C different possible outcomes:

$$\psi : \mathbb{R}^C \to \left\{ \psi \in \mathbb{R}^C \mid \psi_i > 0, \sum_{i=1}^{C} \psi_i = 1 \right\}$$

and is defined as follows:

$$\psi(\boldsymbol{x}_j) = \frac{e^{x_j}}{\sum_{c=1}^{C} e^{x_c}} \quad \forall j \in \{1, \ldots, C\} \tag{2.6}$$

Then, linear regression becomes a classification problem with C classes where the following probability is maximized:

$$P(y_i = c \mid x_i; \boldsymbol{\theta}) = \frac{e^{\left(\theta_c^\top x_i\right)}}{\sum_{j=1}^{C} e^{\left(\theta_j^\top x_i\right)}} \tag{2.7}$$

Chapter 3 makes use of these models for imbalanced binomial classification as applied to peak traffic forecasting. Chapter 4 later extends this to regression (using the same dataset).

2.1.1 Beyond linear models

The softmax function can be applied in principle to any function approximator f transforming regression to a classification by applying ψ at the outputs. For multi-class kernel classification methods such as Support Vector Machines (SVMs) the dominating paradigm is to formulate the problem as multiple binary problems where $C = 2$ and usually extended it as one-vs-rest or all the one-vs-one combinations. However, multi-class extensions exist [103].

Non-linear extensions of linear algorithms can either be done explicitly, or via the kernel trick, which avoids an explicit mapping. To derive a kernelized version of a linear model, I start from the observation that the parameter vector $\boldsymbol{\theta}$ can be expressed as a linear combination of the N training samples:

$$\boldsymbol{\theta} = \sum_{i}^{n} \alpha_i y_i \boldsymbol{x}_i \tag{2.8}$$

where α_i is the number of times x_i was misclassified, also known as the expansion coefficients.

It can now be clearly seen that the main problem with kernel methods, specifically SVMs is that the complexity lies in the number of examples, and

hence working with large datasets can become problematic. Instead of fitting the parameters $\boldsymbol{\theta}$, the vector $\boldsymbol{\alpha}$ is updated:

$$\hat{y} = f(\boldsymbol{\theta}^\top \boldsymbol{x}) \tag{2.9}$$

$$= f\left(\sum_i^N \alpha_i y_i \boldsymbol{x}_i\right)^\top \boldsymbol{x} \tag{2.10}$$

$$= f \sum_i^N \alpha_i y_i (\boldsymbol{x}_i \cdot \boldsymbol{x}) \tag{2.11}$$

Finally, the dot product can be replaced with a kernel K. Mercer's theorem implies that any positive semi-definite matrix K is the Gramian matrix of a set of vectors. A positive semi-definite matrix is a matrix K of dimension N, which satisfies, for all vectors v, the property:

$$v^\top K v = \sum_{i=1}^N \sum_{j=1}^N v_i \, K_{ij} \, v_j \geq 0 \tag{2.12}$$

Then K acts as a feature map Ψ without computing $\Psi(\boldsymbol{x})$ explicitly, yielding the general kernel method:

$$f(\boldsymbol{x}, K) = \sum_{i=1}^N \alpha_i K(x_i, x) \tag{2.13}$$

In practice, however Mercer's condition can be relaxed for matrices K, still resulting in reasonable performance. For Gaussian processes, K is also a covariance function which implies that the Gram matrix K (i.e. kernel) is a covariance matrix. In this thesis I use Gram matices for visualising the activations of several layers in Chapter 5.

2.1.2 Feed forward neural networks

Roughly, neural networks can also be thought of as learning kernels, since a nonlinear mapping is learned in a two layer neural network. Indeed, extensions

exist, where (linear) SVMs are used as pure classifiers, based on features learned from neural networks [165, 186]. However, due to the increased computational cost and the lack of availability of multi-class SVMs, this is not often done in practice very often.

Perceptrons The simplest form of neural network is the Perceptron. Even though in the original formulation, the transfer function was the hard sign function, the extension from linear models is trivial and can be done using the logistic sigmoid function:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) \quad \text{where} \quad \sigma(x) = 1/(1+e^{-x})$$

Multi-layer perceptrons Perceptrons can be trivially extended to multi-class problems via the softmax function (Equation 2.6):

$$h = \sigma(\boldsymbol{\theta}_0^\top \boldsymbol{x}) \tag{2.14}$$

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}) = \psi(\boldsymbol{\theta}_1^\top h) \tag{2.15}$$

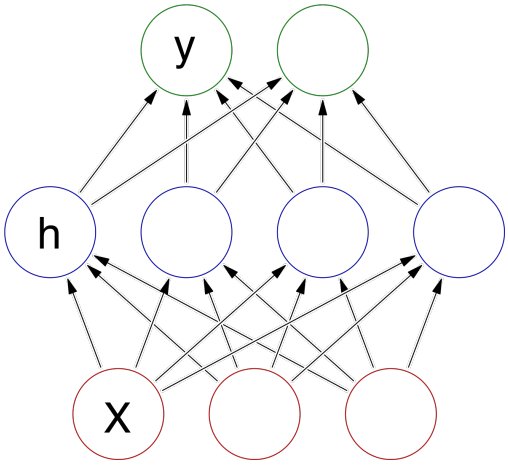


Figure 2.2: A feed-forward neural network with one hidden layer.

Then, the hidden activation h acts as a kernel (but not in a strict definition),

where the representation is learned. This is subsequently passed through a pure linear layer and then passed through the softmax in order to arrive at a multi-class Multi-Layer Perceptron (MLP). The term MLP is a synonym for Feed Forward Neural Network (FFNN - Figure 2.2) when used for classification. Of course, more than one hidden layer can be used, and this amounts to adding more equations producing multiple $h_i \in \{1, \ldots, L\}$ in Equation 2.14 where L are the number of hidden layers.

Residual connections Most likely residual connections [67] were inspired by Highway Networks [157]. In the latter, an additional weight matrix is used to learn the skip weights. This was simplified. The core idea is to enable shortcuts over layers, by bypassing the non-linear transformation with an identity function. This helps alleviate exploding and vanishing gradients. Adding an additional layer and applying a residual connection over the previous example:

$$
\begin{aligned}
h_0 &= \sigma(\theta_0^\top \boldsymbol{x}) \\
h_1 &= \sigma(\theta_1^\top \boldsymbol{x}) + \boldsymbol{x} \\
f(\boldsymbol{x}, \theta) &= \psi(\theta_1^\top h_1)
\end{aligned}
$$

In this case, h_0 will possibly made redundant by skipping the parameters learned in θ_1. Of course, several layers could be skipped at a time. Another extension of the idea is when residual connections exist from all previous layers to all the next layers. These are referred to as DenseNets [78] (idea shown in Figure 2.3).

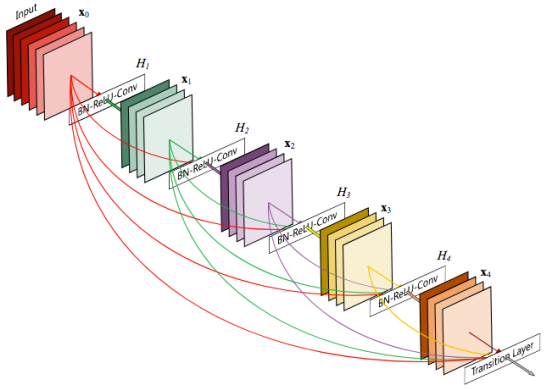


Figure 2.3: DenseNet with a 5-layer dense block with a growth rate of 4. From [78]

2.1.3 Convolutional neural networks

In this section I show that any FFNN dense layer can be converted to a CNN layer. The reverse is also possible but not illustrated here. Firstly, Causal convolutions can be obtained by convolving the signal with a shifted version of itself, or by masking future time steps. This process is illustrated in Figure 2.4 and can be examined in relation to Figure 2.1.

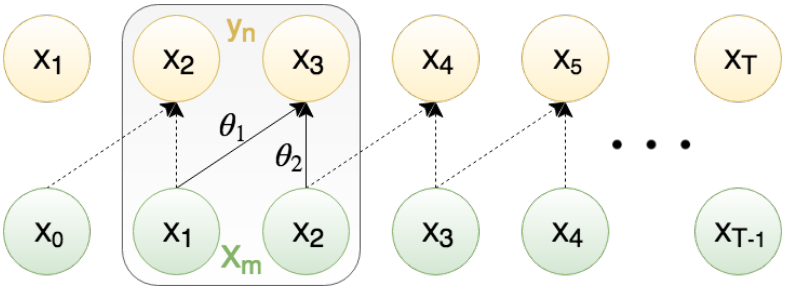


Figure 2.4: A causal convolution with a filter width of 2. The top yellow signal represents the ground truth y values and is a shifted version of $\boldsymbol{x}$.

Discrete convolutions The discrete convolution of two signals f and g is formulated as:

$$\begin{aligned}(f * g)[n] &= \sum_{m=-\infty}^{\infty} f[m]g[n-m] \\ &= \sum_{m=-\infty}^{\infty} f[n-m]g[m]\end{aligned}$$

In the depicted case in Figure 2.4 the parameters θ have support in the set $\{\theta_1, \theta_2\}$ since the filter width $F = 2$ and therefore a finite summation is used:

$$f(\boldsymbol{x}, \theta) = (\boldsymbol{x} * \theta)[n] = \sum_{m=-M}^{M} x[n-m]\theta[m] \tag{2.16}$$

Feed forward dense layers to convolutional layers Any Feed-Forward Neural Network (dense) layer can be converted to a Convolutional Layer since any

convolution can be constructed as a matrix multiplication. The computations can be constructed using a Toeplitz matrix such as A below:

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{bmatrix} \tag{2.17}$$

which is a matrix where $A_{i,j} = A_{i+1,j+1} = a_{i-j}$.

Then, the convolution operation can be constructed as a matrix multiplication where the parameter vector θ for one layer is converted into a Toeplitz matrix. For example, the convolution of θ and $\boldsymbol{x}$ can be written as:

$$y_n = x_t = f(\boldsymbol{x}, \theta) = \theta * \boldsymbol{x} = \begin{bmatrix} \theta_1 & 0 & \dots & 0 & 0 \\ \theta_2 & \theta_1 & \dots & \vdots & \vdots \\ \theta_3 & \theta_2 & \dots & 0 & 0 \\ \vdots & \theta_3 & \dots & \theta_1 & 0 \\ \theta_{m-1} & \vdots & \dots & \theta_2 & \theta_1 \\ \theta_m & \theta_{m-1} & \vdots & \vdots & \theta_2 \\ 0 & \theta_m & \dots & \theta_{m-2} & \vdots \\ 0 & 0 & \dots & \theta_{m-1} & \theta_{m-2} \\ \vdots & \vdots & \vdots & \theta_m & \theta_{m-1} \\ 0 & 0 & 0 & \dots & \theta_m \end{bmatrix} \begin{bmatrix} x_{t-1} \\ x_{t-2} \\ x_{t-3} \\ \vdots \\ x_{t-\Delta} \end{bmatrix} \tag{2.18}$$

Specifically for the case depicted in Figure 2.4 the Causal convolution is obtained by masking $\boldsymbol{x}$ to the observations within the grey box, and referring to the notation in Equation 2.16, $m = 2 = F$ and $n = t - \Delta = 2$. At this point it should be evident that Equation 2.18 is equivalent with Equation 2.5 and can be written compactly as Equation 2.16. In practice, a bias is also commonly used - this is not depicted in Figure 2.4, but one could imagine an extra parameter θ_0

which has an input which is always 1.

Prediction horizon While Figure 2.1 and Figure 2.4 depict prediction horizons equal to one, in order to move the prediction horizon further in time, the only necessary change is to shift the yellow depictions to the right in both figures. The prediction horizon h indicates how far in the future predictions are made, in the case of prediction problems.

Output channels It is usually the case that CNNs change the number of input channels. Then, the parameters θ^c can be repeated as many times as the number of required output channels C_{out} is set to. This is equivalent to repeating the convolution with different learned parameters, as many times as there are output channels. Or another way to imagine the process is to picture several parallel convolutions with the same inputs. Since the learned parameters for each convolution is initialized differently, each will learn a different kernel.

Output observations: Padding & stride An important parameter is the stride, which is labelled only in Figure 2.1. However, Figure 2.4 also depicts a stride of one with the dotted grey box (sliding window). This parameter indicates how many steps to the right the sliding window is moved at one time. For larger strides, the window is moved to the right more than one observation at a time. Padding refers to adding observations (commonly at either the beginning or the end of the time series) in order to change the number of output frames (Equation 2.19. If no padding is used, then the padding is said to be *valid* while if the output is intended to have the same number of frames as the input, the padding is said to be *same*.

Dilated convolutions Dilated convolutions are essentially identical with Causal convolutions with the difference that some of the inputs are skipped. Figure 2.5 shows a dilated convolution with a dilation factor of two. Every second input observation is skipped. The convolution is performed as if the skipped input (Figure 2.5 x_1 - red) does not exist. However, x_1 will be used as an input after the window will be moved to the right for the next operation. Dilated convolu-

tions are typically used for high sampling frequency data such as audio, where a large receptive field Δ is required. Then, multiple such layers as depicted in Figure 2.5 can be stacked on top of each other as it is done in the WaveNet architecture [170].

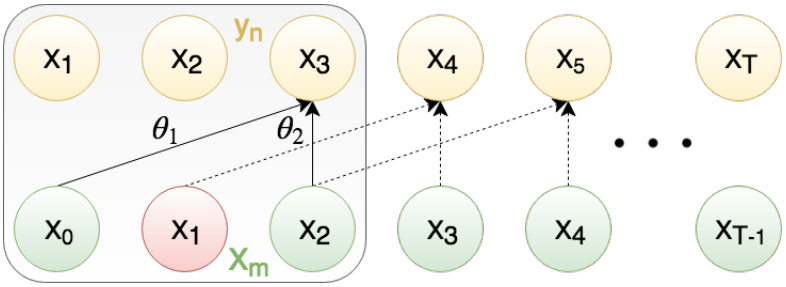


Figure 2.5: Dilated convolution with a dilation factor of two.

In general the number of output frames T_{out} for a one dimensional convolution is given by the following formula:

$$T_{out} = \frac{T_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{filter\_width} - 1) - 1}{\text{stride}} + 1 \qquad (2.19)$$

Pointwise convolutions Pointwise convolutions, also known as 1x1 convolutions or Network in Network are a special case where the filter width is equal to 1 and the only change is the number of output channels. In essence 1x1 convolutions only change the dimensionality of the output. A simpler way to think about depthwise convolutions is to imagine them as a simple feed-forward dense layer where the number of input channels (Figure 2.2 - red) are the number of input neurons and the output channels (Figure 2.2 - blue) are equal to the number of output neurons. This is equivalent to multiplying each observation in $\boldsymbol{x}$ with a set of parameters θ that change the dimensionality.

Grouped convolutions Convolutions can be grouped. When the number of groups is set to 2 for example, one can think of two separate convolutions being performed in parallel. However each convolution will have access to only half the input channels and will also produce only half the the output channels.

Finally, the outputs will be concatenated to produce the total requested number of channels.

An extreme case of grouped convolutions is when the number of groups is set equal to the number of input channels. In this case, each input channel is convolved with its own set of filters of size C_{out}/C_{in}. This case is termed in literature as a depthwise separable convolution.

A standard convolution filters and combines inputs into a new set of outputs in one single step. In depthwise separable convolutions, each channel is first convolved with its own set of filters, then a 1x1 convolution is applied in order to combine the outputs.

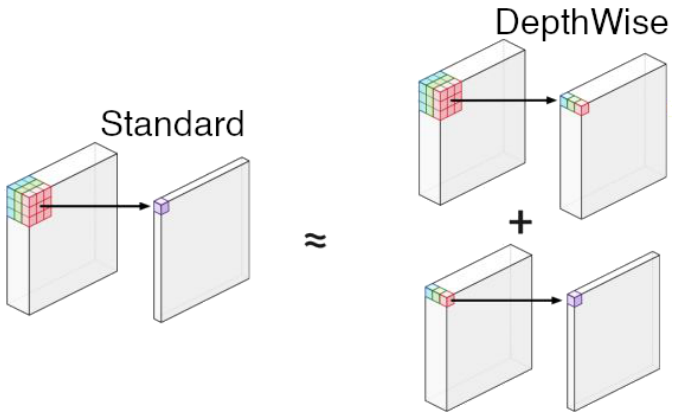


Figure 2.6: 2D Depthwise separable convolution (right) vs. standard (left).

In some cases, grouped convolutions happen to work better in practice since filter relationships are sparse. When the number of output channels is large and the kernel size is also large, the computational cost can be significantly reduced. The number of multiplications (or the number of parameters is compared in the equation below, where C represents channels, W is the width of the kernel and V is the output volume or the number of output frames:

$$\frac{\text{Standard convolution}}{\text{Depthwise separable conv}} = \frac{C_{in} \times V \times (W + C_{out})}{C_{in} \times V \times W \times C_{out}} = \frac{W + C_{out}}{W \times C_{out}} \tag{2.20}$$

It can be seen that when setting $W = 2$ and $C_{out} = 128$, the depthwise separable convolution has twice fewer parameters. If the filter width is increased to

$W = 3$ then it will have 66% fewer and so on.

The idea was originally introduced in the AlexNet [97] architecture as an engineering feat to speed up training and recently, it has also been applied to low power mobile deep learning vision applications [75]. Parallels to the Inception model [163] have also been made in the Xception model [33] which generalizes the concept. The architectural concept has also proven to be a key component in the MultiModel [82] architecture which consists of a single neural network model that can simultaneously learn multiple tasks from various multiple domains.

2.1.4 Autoregressive Generative CNNs

Having discussed most of the ingredients in subsection 2.1.3 here I start by detailing the WaveNet architecture. I finally provide a diagram that explains in parallel both the time unrolled view of the architecture, as well as the where the dilated convolutional blocks are located, providing an overall view of the number of frames, channels (dimensions / neurons) and the convolutions. I also explain how the number of frames and the number of channels change between layers. Then I explain the structure of each dilated block in WaveNet and finally put everything together, explaining the skip connections and how to condition the network with other signals.

Frames and channels In Figure 2.9 the receptive field Δ has 8 frames, consisting of $\{\boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_{t-\Delta}\}$. This will produce one output $\hat{\boldsymbol{x}}_t$. In general, for an input of T frames, there will be $T - \Delta + 1$ output frames.

For raw mono audio, the input channels can either be: 1 or 256 if the audio is quantized using μ-Law companding and then subsequently encoded to a one-hot 256 valued vector). μ-Law companding only changes the bit depth of the audio from $2^{16} = 65536$ which is standard audio quality to $2^8 = 256$ which reduces the number of possible outputs and makes training possible with the cross-entropy loss.

Dilated convolutions One core ingredient is the dilated convolution. Multiple of these are stacked on top of each other, where each layer has a increased dilation rate. In Figure 2.9 I also display the dilated block architecture in order to exemplify how everything in the architecture is put together. This is for a plain architecture without any conditioning.

The first input layer is a convolutional layer with a filter width of F. In fact all convolutions, except for the output layers have a filter width of F. The purpose of this layer is to change the number of input channels C_{in} from whatever the input dimensionality in $\boldsymbol{x}_{<t}$ is to the number of channels in each dilated block C_h. In addition, the input layer changes the number of input frames, according to Equation 2.19 (in Figure 2.9 from 8 to 7).

$$\underbrace{\boldsymbol{x}^1}_{C_h} = tanh(\boldsymbol{\theta}_{in} * \underbrace{\boldsymbol{x}_{<t}}_{C_{in}}) \tag{2.21}$$

This is followed by a series of dilated block layers (only 2 in Figure 2.7) which all have C_h channels. Every dilation block also changes the number of frames according to Equation 2.19, reducing them until there is only one frame at the top of the block-layers stacks. This is the main purpose of the dilated blocks - to grow very large receptive fields by stacking dilated convolutions on top of each other, with an ever increasing dilation rate (however the dilation pattern e.g. 1, 2, 4, 8, etc. can be repeated multiple times). There is also no restriction on how the dilation rate increases, it does not necessarily have to be increasing powers of two.

Inside the dilated blocks Each dilated block is composed of two initial convolutions, side by side, which take exactly the same inputs from the previous layer $\boldsymbol{x}^{\ell-1}$. The first one goes through a *tanh* - τ activation while the second one goes through a *sigmoid* - σ activation. This mechanism is most likely inspired by recurrent neural networks. In essence, the sigmoid acts like a gating mechanism that can inhibit the main tanh signal. This is a good approach under the plausible hypothesis that each layer learns coefficients for a filter bank and thus some frequencies should be cut off.

The convolutions for τ and σ are not depicted in Figure 2.7 as two green

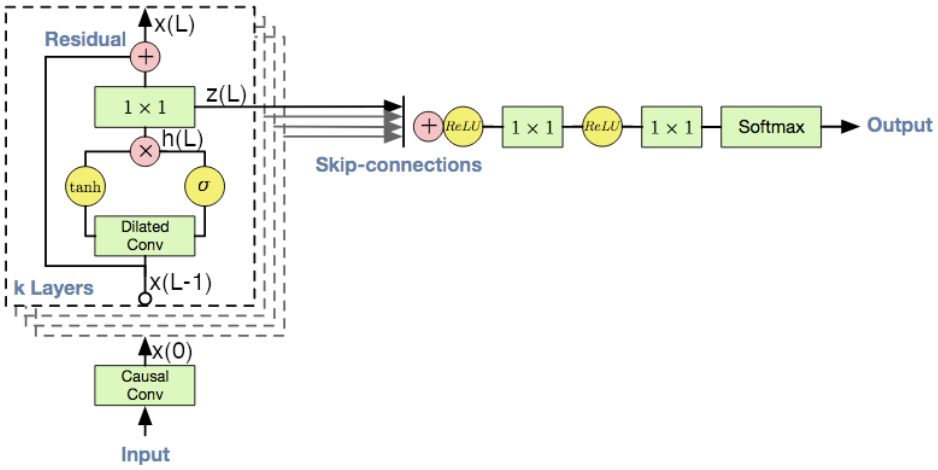


Figure 2.7: The dilation block that is repeated every dilation layer, is depicted inside the dotted line (these are also shown in Figure 2.9). This depicts the operations in each dilated block, and also shows the last two output layers. Adapted from [170].

boxes since in practice the computations can be done in parallel in one single convolution. However it should be understood that each yellow circle corresponds to it's own separate convolution.

In the case where conditioning is applied, both convolutions get summed with another convolution which is parametrized by $\boldsymbol{V}$ where the conditioning signal is $\boldsymbol{y}^{\ell-1}$. The latter is either a scalar or a vector of the same length as $\boldsymbol{x}^{\ell-1}$ and can be obtained with the same procedure described in Figure 2.9 for the main signal:

$$\underbrace{\boldsymbol{x}^{\ell}}_{C_h} = \underbrace{\boldsymbol{x}^{\ell-1}}_{C_h} + \overbrace{\boldsymbol{\theta}_r^{\ell} \cdot \underbrace{\boldsymbol{h}^{\ell}}_{C_{res}}}^{C_h}$$

$$\boldsymbol{h}^{\ell} = \tau\left(\boldsymbol{W}_f^{\ell} * \boldsymbol{x}^{\ell-1} + \boldsymbol{V}_f^{\ell} * \boldsymbol{y}^{\ell-1}\right) \odot \sigma\left(\boldsymbol{W}_g^{\ell} * \boldsymbol{x}^{\ell-1} + \boldsymbol{V}_g^{\ell} * \boldsymbol{y}^{\ell-1}\right) \tag{2.22}$$

In SynthNet, the convolutions in Equation 2.22 are changed to grouped convolutions, and in combination with the pointwise $1x1$ convolution, this effectively results in what is called a depthwise separable convolution. This is not used in either the original WaveNet [170] nor the latter DeepVoice [9] derivative.

Skip connections Figure 2.9 depicts a standard neural network architecture where the output of each layer goes to the next and so on. However, in addition a similar idea as in DenseNets [78] is used.

There is an additional output from each dilated block in addition to $\boldsymbol{x}^\ell$ and this is where the outputs towards the softmax come from. These are not present in the SynthNet architecture presented in Chapter 5 which uses only the top most output.

In essence, the output before the residual connection is sent towards the outputs, from each layer:

$$\underbrace{\boldsymbol{z}^\ell}_{C_h} = \boldsymbol{\theta}_r^\ell \cdot \underbrace{\boldsymbol{h}^\ell}_{C_{res}} \tag{2.23}$$

however before going through this convolution, only the last frame is kept, so all the $\boldsymbol{z}^\ell$ consist of only one frame. All of these are finally stacked into one matrix and the matrix is reduced to a vector of 1 frame and C_h channels, by summing over the layer dimension (Figure 2.9 red plus circle):

$$\boldsymbol{z}_{all} = relu([\boldsymbol{z}^1\ \boldsymbol{z}^2\ \ldots\ \boldsymbol{z}^L]^\top) \tag{2.24}$$

In other, words, all the layer outputs are collapsed onto one. Other means of performing this reduction is possible, for example by concatenating all frames (but then C_h would have to be a much smaller number) by performing a 2D convolution over the $\boldsymbol{z}^\ell$ stack, etc.

This is then followed by two other $1x1$ convolutional layers that reduce the number of channels, up to the softmax (these are not displayed in Figure 2.9, but are visible in Figure 2.7):

$$\boldsymbol{z}_{out} = relu(\boldsymbol{\theta}_{all} \cdot \boldsymbol{z}_{all}) \tag{2.25}$$

$$\hat{\boldsymbol{x}}_t = softmax(\boldsymbol{\theta}_{out} \cdot \boldsymbol{z}_{out}) \tag{2.26}$$

Generation At generation time, autoregressive models use the output of the previous time steps in order to fill up the receptive field and generate the next output. The process is repeated until a sufficient number of frames is generated and is much slower than training. Both processes are shown in Figure 2.8.

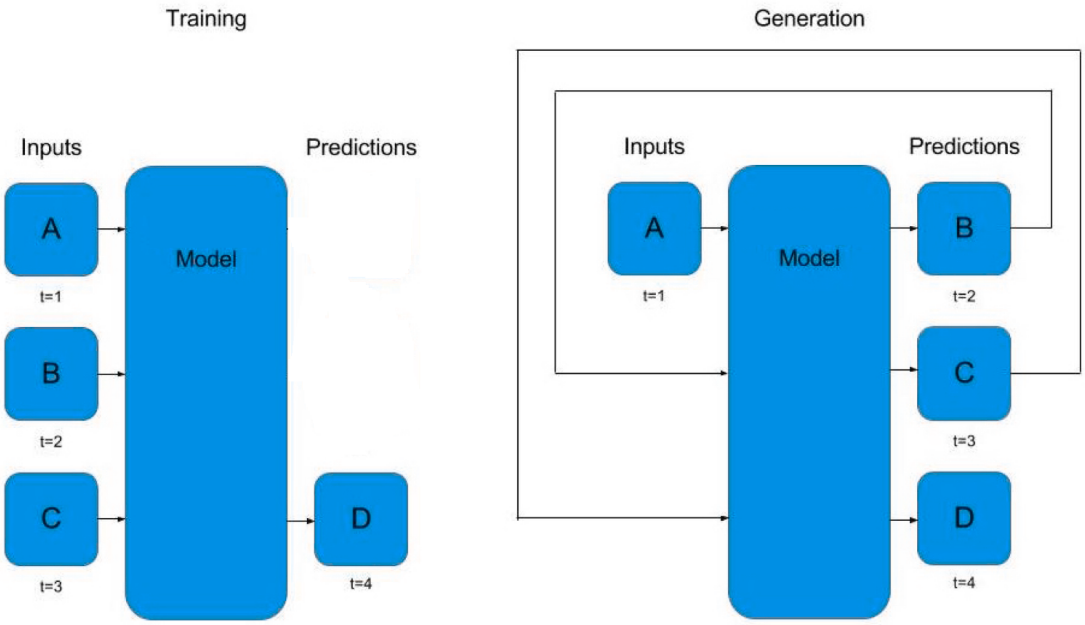


Figure 2.8: Left: training an autoregressive model. Right: generation by repeated sampling. In practice, padding can be added at generation time before the first actual sample is generated.

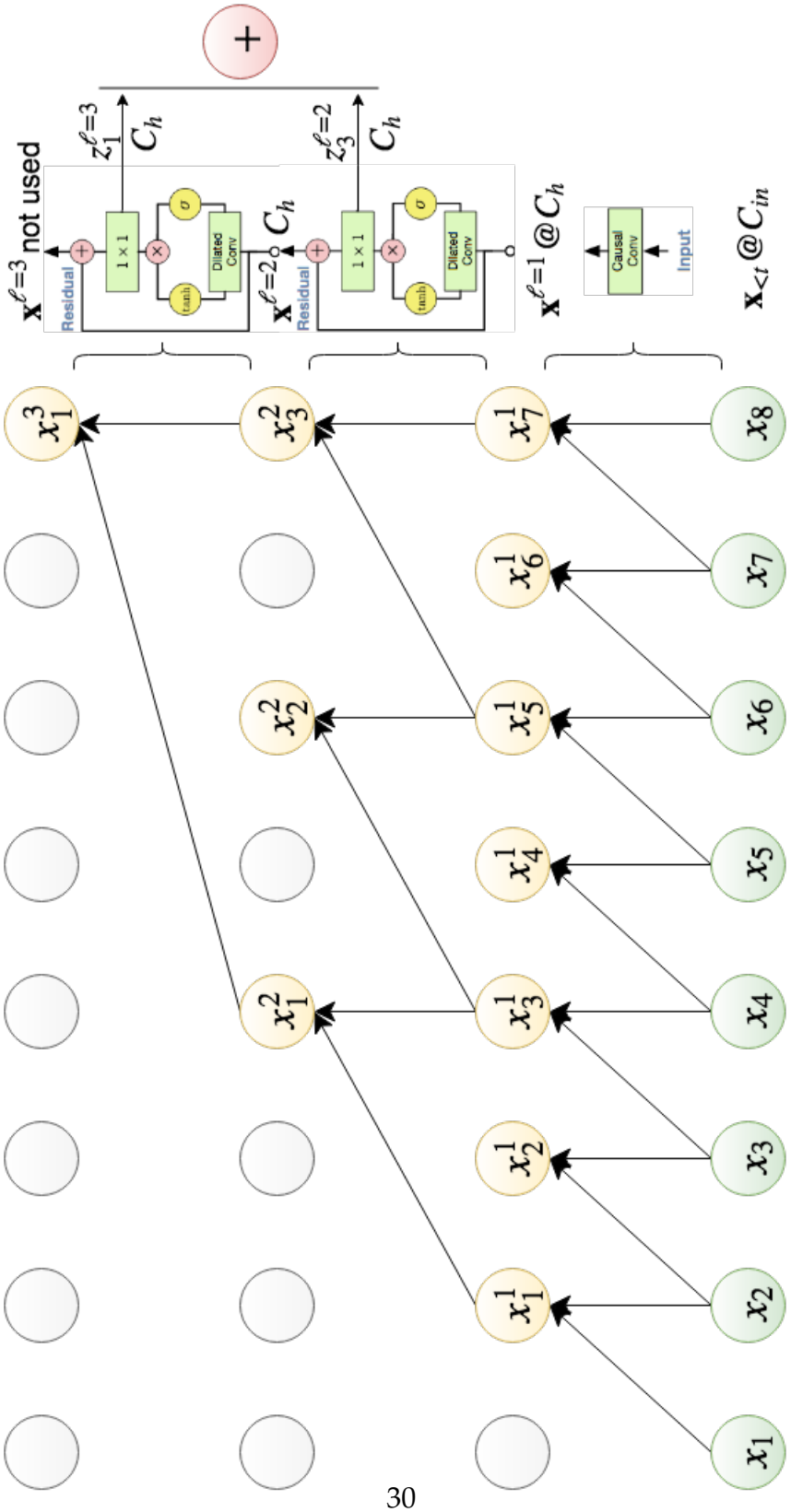


Figure 2.9: Multiple dilated convolutional layers are stacked. This results in a very large receptive field, which is useful for high bandwidth data such as audio.

2.2 Multi-task learning

Multi-task learning, as the name suggests, refers to learning multiple (usually related) tasks in parallel or using auxiliary tasks in order to increase overall performance. MTL has many successful machine learning applications, ranging from speech recognition, computer vision, drug discovery to natural language processing. 2.4.2 discusses recent advances in MTL and also provides an overview of MTL methods in the context of deep learning.

I define MTL in the most general sense as possible: given S tasks $\{(\boldsymbol{x}_i, y_i)_{i=1}^{N_1}, (\boldsymbol{x}_i, y_i)_{i=1}^{N_1}, \ldots, (\boldsymbol{x}_i, y_i)_{i=1}^{N_S}\}$ each with a possibly different number of observations N_S and that are either related or not, the goal is to solve problems of the form $f^s : X^s \rightarrow Y^s, \ \forall s$. Each task can be based on completely different data that can be of different types and can have different distributions. The core idea of MTL is to solve these multiple tasks at once, and by doing so hopefully do better overall by transferring knowledge learned on one task to improve other tasks, and vice versa (i.e. leverage task relatedness). Sometimes, additional tasks which are not of importance (i.e. auxiliary) are added in order to improve the performance of one particular task.

This chapter will demonstrate that solving multiple linear regression problems of related tasks is equivalent to solving one large multi-task regression problem and *vice versa*.

Vector autoregression and multiclass classification VAR is a special case of MTL, as is multiclass classification. For vector autoregression and keeping the multivariate regression notation (see section 2.1, Equation 2.2 to Equation 2.5) there are problems of the form $f : X \rightarrow Y \subseteq \mathbb{R}^S$ where each series s $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots \boldsymbol{x}^s$ has N observations, $X \in \mathbb{R}^{N \times S}$ and is parametrized by $\boldsymbol{\theta}^s$ (and for VAR $\boldsymbol{\theta}_{\Delta}^s$). The tasks can be related and the relationship can be described using a given matrix A or learned from the data. More on that later on. To summarise, VAR is a special case of MTL where the input space is identical for all tasks.

Furthermore, multiclass classification is also a special case of VAR where each class is a task and all tasks have the same inputs $f : X \rightarrow Y \in \{1, 2, \ldots, S\}$ where $\boldsymbol{x}_i \in X$ and $y_i \in \{1, 2, \ldots, S\}$ is usually one-hot encoded.

In the next subsection I start with a generic definition of multi-task learning for linear models and the squared error. I also initially focus on the special VAR case since having the same number of examples for all tasks can lead to a cleaner notation. Then I discuss various regularization techniques and finally generalize beyond linear models to regularization for neural networks.

Solving a linear MTL problem Since Chapter 4 makes use of VAR as a special case of MTL, I start as in section 2.1 with linear models:

$$f^s(\boldsymbol{x}, \boldsymbol{\theta}^s) = \boldsymbol{\theta}^{s\top} \boldsymbol{x} \tag{2.27}$$

The goal is to find the parameters θ that minimize the empirical error. This can be done by taking the sum of the mean squared error for each task:

$$\min_{\boldsymbol{\theta}^1,\dots,\boldsymbol{\theta}^s} \sum_{s=1}^{S} \frac{1}{N_s} \sum_{i=1}^{N_s} (\boldsymbol{\theta}^{s\top} \boldsymbol{x}_i^s - y_i^s)^2 \tag{2.28}$$

Then, for VAR the notation can be further simplified and written more compactly as exemplified in Equation 2.29. This is possible since for VAR all series are the same length and can be combined into one single matrix X (which is constructed as depicted in Figure 2.1). This implies that all tasks are fed all the series and that $\boldsymbol{y}_i$ is the output vector at example i, for all tasks.

$$\min_{\boldsymbol{\theta}^1,\dots,\boldsymbol{\theta}^s} \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} (\boldsymbol{\theta}^{\top} \boldsymbol{x}_i - y_i^s)^2 \tag{2.29}$$

$$=$$

$$\min_{\Theta} \frac{1}{N} \| \underbrace{X}_{N\times\Delta} \underbrace{\Theta}_{\Delta\times S} - \underbrace{Y}_{N\times S} \|_{\mathrm{F}}^2 \tag{2.30}$$

At this point it is also a good idea to define the squared Frobenius norm, for clarity in the next paragraphs:

$$\|A\|_{\mathrm{F}}^2 = \mathrm{Tr}(A^{\top} A) = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2 = \sum_{i=1}^{\min\{m,n\}} \sigma_i^2(A) \tag{2.31}$$

where $\sigma_i(A)$ are the singular values of A and the trace Tr is the sum of diagonal entries of a square matrix. An interesting property of the Frobenius norm is that it is invariant under rotations.

2.2.1 Regularization for multi-task learning

A good starting point is to simply add an ℓ_2 loss term for each task. This however, does not produce any task dependencies:

$$\min_{\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^S} \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} (\boldsymbol{\theta}^{s\top} \boldsymbol{x}_i - y_i^s)^2 + \lambda \sum_{s=1}^{S} \|\boldsymbol{\theta}^s\|_2^2 \tag{2.32}$$

$$=$$

$$\sum_{s=1}^{S} \Big(\min_{\boldsymbol{\theta}^s} \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{\theta}^{s\top} \boldsymbol{x}_i - y_i^s)^2 + \lambda \|\boldsymbol{\theta}^s\|_2^2 \Big) \tag{2.33}$$

$$=$$

$$\min_{\Theta} \frac{1}{N} \|X\Theta - Y\|_{\mathrm{F}}^2 + \lambda \|\Theta\|_{\mathrm{F}}^2 \tag{2.34}$$

The regularization term can be rewritten using Equation 2.31 and since the trace is invariant under cyclic permutations, and an identity matrix can be added like so:

$$\|\Theta\|_{\mathrm{F}}^2 = \mathrm{Tr}(\Theta^\top \Theta) = \mathrm{Tr}(\Theta I \Theta^\top) \tag{2.35}$$

This can be further rewritten as $A = I$:

$$\begin{aligned}\operatorname{Tr}(\Theta A \Theta^\top) &= \sum_{j=1}^{\Delta} \boldsymbol{\theta}_j^\top A \boldsymbol{\theta}_j \\ &= \sum_{s=1}^{S} \sum_{k=1}^{S} \sum_{j=1}^{\Delta} a_{s,k} \theta_j^s \theta_j^k \\ &= \sum_{s=1}^{S} \sum_{k=1}^{S} a_{s,k} \sum_{j=1}^{\Delta} \theta_j^s \theta_j^k \\ &= \sum_{s=1}^{S} \sum_{k=1}^{S} a_{s,k} \, \boldsymbol{\theta}^{s\top} \boldsymbol{\theta}^k\end{aligned}$$

It is easy to observe now, that if $A = I$ the diagonal elements are one when $s = k$ and the rest are zero. This is consistent with no task dependency since there is a weight of 1 for each task, while the weights for all other tasks are 0. Equation 2.32 corresponds to setting $A = I_S$ where each task is regularized individually and is essentially identical to using the same input for all tasks and applying a regularizer for each task.

In order to induce task relatedness, MTL regularization amounts to weighting the task relatedness which can be described by a positive (semi)definite matrix A. The full compact notation for the general task relatedness regularization term (based on the squared loss) is:

$$\min_{\Theta} \frac{1}{N} \|X\Theta - Y\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Theta A \Theta^\top) \tag{2.36}$$

which also has a nice gradient:

$$\nabla L(\Theta) = \frac{2}{N} X^\top (X\Theta - Y) + 2\lambda \Theta A \tag{2.37}$$

showing that Equation 2.36 can be extended to other loss functions as well.

MTL regularization using graphs A special case of MTL regularization is when the graph structure of the tasks is known. For most MTL problems, there are usually groups of tasks and hence there exist multiple clusters at the out-

puts. The adjacency matrix can be used to enforce dependency between tasks via regularization by taking $A = L + \lambda I$ where L is the graph Laplacian of the adjacency matrix:

$$A = L + \lambda I = \sum_{s=1}^{S} \sum_{k=1}^{S} a_{s,k} \|\boldsymbol{\theta}^s - \boldsymbol{\theta}^k\|^2 + \lambda \|\boldsymbol{\theta}^s\|^2 \tag{2.38}$$

however A is given and not inferred from the data.

Learning the task relationship matrix A If the matrix describing the relationships between tasks is not known, it is possible to learn it by adding an additional penalty term Ω and alternating between Equation 2.41 - fitting the parameters Θ while fixing $A = A_*$ and Equation 2.42 - finding the task relatedness matrix A where $\Theta = \Theta_*$ is fixed:

$$\min_{\Theta, A} \frac{1}{N} \|X\Theta - Y\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Theta A \Theta^\top) + \gamma \Omega(A) \tag{2.39}$$

$$\text{alternate } \Downarrow \text{ following eqs.} \tag{2.40}$$

$$\min_{\Theta} \frac{1}{N} \|X\Theta - Y\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Theta A_* \Theta^\top) \tag{2.41}$$

$$\min_{A} \lambda \operatorname{Tr}(\Theta_* \, A \Theta_*^\top) + \gamma \Omega(A) \tag{2.42}$$

One possible choice for Ω is $\operatorname{Tr}(A^{-2})$. Looking at the Frobenius norm (Equation 2.31) and the output space rotation trick in paragraph 2.2.1 this amounts to $\gamma \sum_s (1/\sigma_s^2)$. This ensures that task importance will not vanish for any task.

MTL regularization computations Assuming that the matrix A in Equation 2.36 is symmetric positive (semi)definite, then it is possible to diagonalize the matrix via singular value decomposition (SVD) $A = U\Sigma U^\top$ where $\Sigma = diag(\sigma^1, \sigma^2, \dots \sigma^s)$. Then, setting $\tilde{\Theta} = \Theta U$ and $\tilde{Y} = YU$ it is possible to rewrite Equation 2.36 as:

$$\min_{\Theta} \frac{1}{N} \|X\tilde{\Theta} - \tilde{Y}\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\tilde{\Theta} \Sigma \tilde{\Theta}^\top) \tag{2.43}$$

where the regularization term is reduced to a sum of product norms weighted by the diagonal elements σ. The above can be uncompressed to:

$$\sum_{s=1}^{S} \left(\min_{\boldsymbol{\theta}^s} \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}_i^s - \tilde{\theta}^{s\top} \boldsymbol{x})^2 + \lambda \sigma^s \|\boldsymbol{\theta}^s\|^2 \right) \tag{2.44}$$

where $\Theta = \tilde{\Theta} U^\top$. However, this is almost identical to Equation 2.32 with the exception that now the output space is rotated so that the task relationship matrix becomes diagonal and there is a weight σ^s that depends on all the other tasks. This clearly shows that task regularization depends on the output space, as well as the parameter covariance between tasks.

However, since the Frobenius norm Equation 2.31 is rotation invariant, the regularization term does not change. This means that the computations can also be done in parallel, on a per-task basis instead of in one single model, with the additional cost of computing σ^s for every iteration.

Implications for multiclass classification For multiclass classification, the entries in the matrix Y can be encoded as a one-hot vector, where $|y_i| = C = S$. Solving Equation 2.36 for multiclass problems, even for different loss functions using for example gradient descent, essentially corresponds to a one-vs-all classification problem where A is the identity matrix. In addition, class relatedness can also be encoded via the matrix A, however paragraph 2.2.1 shows that in this case there won't be much of an improvement since rotating the output space for classification does not produce any changes (since it's one-hot).

However, affine transformations in the input space can be applied and are usually the norm. For image multiclass classification (e.g. ImageNet) the images are rotated, translated, cropped, mirrored, noise added and so on. While these transformations are generally considered as a way of augmenting data, here the added perspective is that these affine transformations also act as a regularizer.

Task groups and input sparsity paragraph 2.2.1 shows that MTL problems can be parallelized. However, in the case where the tasks are sparsely grouped (such as in Chapter 4), the efficiency can be further improved by limiting the

dimensionality of the input space, based on the the first order graph connections A where G is the adjacency matrix:

$$a^{ij} = \begin{cases} 1, & i = j; \\ 0, & j \notin G^1(i); \\ 1, & j \in G^1(i). \end{cases}$$

Then, $A \in \mathbb{R}^{S \times S}$, $a_{i,j} \in \{1, 0\}$ can be used to induce sparsity in the input space. This is detailed in subsection 4.3.1, where A is constructed based on the topological structure of road segments. As an extreme case, in order to reduce Equation 2.29 to S single tasks that do not share any information, the matrix $A = I_S$ can be set to the identity matrix.

The idea is to eliminate most of the input space for one task:

$$\min_{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s} \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} (\boldsymbol{\theta}^{s^\top} \boldsymbol{x}_i \odot [(\boldsymbol{a}^s)_{\times \Delta}] - y_i^s)^2 + \lambda \sum_{s=1}^{S} \|\boldsymbol{\theta}^s\|_2^2 \tag{2.45}$$

where $[(\boldsymbol{a}^s)_{\times \Delta}]$ is the binary row from matrix A for task s, repeated Δ times. Here, Δ is the input dimension for one task.

To do this for all tasks, the input space is repeated for each task: $\tilde{X} = \begin{bmatrix} X^1 & X^2 & \ldots & X^S \end{bmatrix}$ and I define $A^s = \left[(\boldsymbol{a}^s)_{\times \Delta} \right] \in \mathcal{R}^{N \times D = (\Delta \cdot S)}$ and $\tilde{A} = \begin{bmatrix} A^1 & A^2 & \ldots & A^S \end{bmatrix}$:

$$\min_{\Theta} \frac{1}{N} \| \underbrace{A}_{N \times D} \odot \underbrace{\tilde{X}}_{N \times D} \underbrace{\Theta}_{D \times S} - \underbrace{Y}_{N \times S} \|_{\mathrm{F}}^2 + \Omega(\Theta) \tag{2.46}$$

If Equation 2.46 is interpreted as a nearest neighbour data selector based on matrices A for each task, and all datasets are combined into one, then some of the features will overlap. However, this is beneficial when using the trace norm, as discussed in subsection 2.4.2.

Learning input sparsity If the priors for designing the matrix A are not known, then sparsity can be enforced automatically based on the ℓ_1 norm which is also known as Least Absolute Shrinkage and Selection Operator (LASSO) method.

In this case, the combination leads to group lasso. In certain cases, the assumption can be that certain dimensions from the inputs are not required. In this case, the ℓ_2 norm is computed across each input dimension d across all tasks, after which sparsity can be enforced on the concatenated vector of ℓ_2 norms:

$$\min_{\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^s} \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} (\boldsymbol{\theta}^{s\top} \boldsymbol{x}_i - y_i^s)^2 + \lambda_1 \left\| \lambda_2 \|\boldsymbol{\theta}_1\|_2^2 \ \ldots \ \lambda_2 \|\boldsymbol{\theta}_\Delta\|_2^2 \right\|_1 \tag{2.47}$$

Beyond linear models Extensions to nonlinear models are trivial via the same methods exemplified in subsection 2.1.1. Setting $f^s = \boldsymbol{\theta}^{s\top} \Phi(\boldsymbol{x})$ where $\Phi(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x}))$ simply amounts to setting $X = \Phi$ in Equation 2.36.

$$\min_{\Theta} \frac{1}{N} \|\Phi\Theta - Y\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Theta A \Theta^\top)$$

In the case of SVMs a kernel function K parametrised by Ψ will result in $f^s = \sum_i K(\boldsymbol{x}, x_i) \psi_i^s$:

$$\min_{\Theta} \frac{1}{N} \|K\Psi - Y\|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Psi A \Psi^\top)$$

Generalized MTL regularization Going back to Equation 2.28 and adding task regularization amounts to:

$$\min_{\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^s} \sum_{s=1}^{S} \frac{1}{N_s} \sum_{i=1}^{N_s} (\boldsymbol{\theta}^{s\top} \boldsymbol{x}_i^s - y_i^s)^2 + \lambda \sum_{s=1}^{S} \sum_{k=1}^{S} a_{s,k} \, \boldsymbol{\theta}^{s\top} \boldsymbol{\theta}^k \tag{2.48}$$

This can also be rewritten in compact form:

$$\min_{\Theta} \frac{1}{N} \|(\underbrace{X}_{N\times\Delta} \underbrace{\Theta}_{\Delta\times S} - \underbrace{\overline{Y}}_{N\times S}) \odot \underbrace{M}_{\Delta\times S} \|_{\mathrm{F}}^2 + \lambda \operatorname{Tr}(\Theta A \Theta^\top) \tag{2.49}$$

where Δ is the maximum dimensionality of all tasks, $N = \sum_s N_s$ is the overall total number of examples over all tasks. Then, $\overline{Y}$ is all zeros except for the target value for each task example, and acts as a mask for data points that do not exist (since not all tasks have the same number of examples) and the matrix

$M \in \{0, 1\}$ is mask that indicates the existing feature space for each task (since not all tasks have the same number of features).

2.2.2 Multi-task learning in neural networks

Multi-task learning for neural networks can be slightly more nuanced. Here, the discussion changes to learned representations. Essentially, in the context of NNs, MTL improves generalization by sharing the learned representations. As discussed in subsection 2.4.2 there are two ways in which parameters are shared: hard and soft parameter sharing. The advantage of NNs and deep learning for MTL comes from the flexibility to data modality and variety of examples and the ability to learn joint representations.

The typical deep learning architecture for MTL can be divided into two parts: the input and output layers, which can consist of any type of architecture, whether designed for vision or for audio or other modalities, and the shared layers, where the connection to the entirety of this section can be made. Since each task imposes it's own constraints on the shared parameters, this prevent overfitting. In it's simplest form, the extension of 2.36 to a neural network with one hidden layer is:

$$\min_{\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^s} \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} (\phi(\boldsymbol{\theta}^s_{out}, \phi(\boldsymbol{\theta}^s_{in}, \boldsymbol{x}_i)) - y^s_i)^2 + \lambda \operatorname{Tr}(\Theta_{out} A \Theta^\top_{out})$$

where $\phi(x) = 1/(1 + e^{-x})$ is the sigmoid activation function. Here, parameter sharing is enforced on the first set of layers Θ_{in} which can consist of virtually any combinations of architectures, and the task regularization is applied at the output layer parameters. This is however only a proof of concept. Figure 2.10 shows a more complicated architecture that uses state of the art bells and whistles, which are combined and designed to induce sparsity:

Essentially in [82] convolutional, attentional and mixtures of experts architectural blocks are combined, as to share as many parameters as possible, into a single deep learning model which jointly learns several large-scale tasks from multiple domains. MTL approaches to deep learning are further discussed in

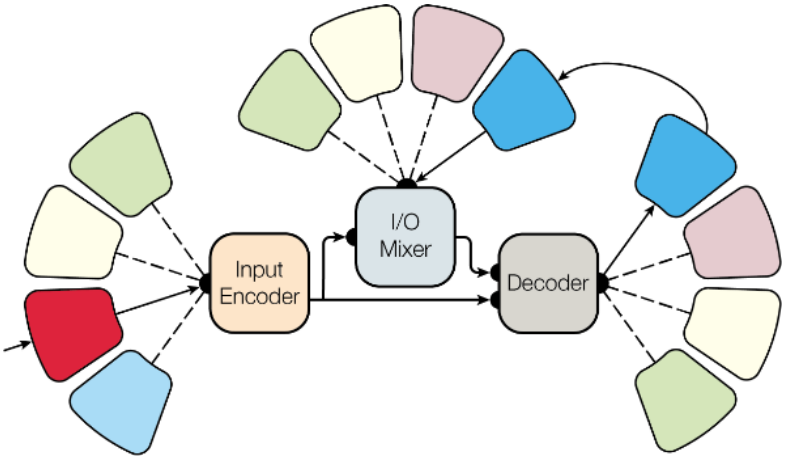


Figure 2.10: Multi-task learning with deep neural networks. Shared representations are learned in both the encoder and the autoregressive decoder. Each colour represents a different modality / task. Taken from [82].

subsection 2.4.2.

2.3 Model fitting with gradient based methods

In this chapter I give an introduction to gradient descent methods that will be used throughout this thesis. Gradient descent methods are very popular for model fitting and are used in virtually all neural network / deep learning research. This is because if the computational tractability and, arguably, the simplicity of the algorithm.

There are many ways to categorise gradient descent optimization methods. For one, there are first order and second order methods, where first order methods are based on the matrix of partial derivatives (Jacobian) while second order methods use the second order derivatives - the Hessian matrix. While the convergence rate of second order methods is higher, the increased computational cost does not usually make them a popular choice in practice, especially for models with a large number of parameters. Another way to characterise gradient descent is based on how much data the loss (and gradient) is computed for, at one time: batch, stochastic (online) and mini-batch gradient descent. The latter is a combination of the first two, as a good compromise between speed and quality.

2.3.1 First order methods

Gradient descent minimizes an objective function $L(\theta)$ where $\theta \in \mathcal{R}$ are the model's parameters. The loss function $L(y, \hat{y})$ measures the agreement of the model's outputs $\hat{y}$ to the ground truth y and is therefore a direct measurement of how well the model's parameters are fitting the input data. The gradient of the objective function $\nabla_\theta L(\theta)$ w.r.t. the model's parameters θ dictates the direction in which the parameters will be updated (in negative gradient direction):

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t)$$

For vanilla methods, while the gradient indicates the direction in which the function is steepest, it gives no information on how far along this direction the update step should be. The step size is called the learning rate η and is one of the most important parameters in gradient descent. If the learning rate is

too small, the optimization process will be very slow. Conversely, if it is too big, taking large steps will result in chaotically updating the parameters in the optimization space.

Batch Gradient Descent In batch gradient descent, the gradient is computed for the entire dataset at once, then the parameters are updated, then the process is repeated until convergence. One pass through the dataset is considered one epoch. This method is guaranteed to approach a global minimum for convex error surfaces and a local minimum for non-convex ones. It is also, however, impractical since it is usually not the case that the entire data set will fit in memory. Furthermore, since the update is done for all data points simultaneously, especially without any regularization, there are risks that the model will overfit the training data. However, this method is very stable (arrives consistently at the same solution) and can be very fast if the data fits in memory, since fewer updates have to be made (usually not more than 10). A typical learning rate is $\eta \in 0.1, 0.9$.

Stochastic Gradient Descent Conversely, in stochastic gradient descent (SGD) the gradients are updated after seeing only one data point. It is also called Online since the model can be conveniently updated at a later time, if new data points arrive. The name stochastic comes from the fact that the order in which the data points are presented every epoch is randomly chosen. In other words, the dataset is shuffled every epoch. Since there are frequent updates, (one per data point) the variance between updates is high. This is because each data point indicates a different (and low-perspective) direction of the gradient. While this might initially seem as a bad idea, the model will eventually average itself to the combined gradient of all data points. This is why a low learning rate $\eta \in 10^{-5}, 10^{-1}$ is used in practice for SGD as opposed to batch gradient descent. Furthermore, since the gradient is stochastic, SGD explores the error surface, 'by accident'. It can also be said that SGD has an inherent self-regularizing property. While this can create problems towards the end of the optimization procedure (since the gradient will be too 'noisy') a typical approach is to gradually decrease the learning rate η in time - this is called learning rate decay and

is also known as Simulated Annealing. The latter meta-heuristic was inspired from annealing in metallurgy. The major disadvantage of SGD is that updating once per parameter can be slow since it is a high speed, low volume process.

Mini-batch gradient descent combines batch with stochastic gradient descent, thus overcoming both previous methods' pitfalls. By using more examples at once, the variance of the parameter updates is reduced, which can possibly make the convergence more stable. It also increases the volume of the gradient updates since computations can be again done simultaneously for multiple data points. The size of the mini-batch is another key parameter that has to be tuned in conjunction with the learning rate, since both affect the noise levels in the gradient updates. Currently, mini-batch gradient descent is ubiquitous and the term SGD is usually also used to refer to mini-batch. Asynchronous SGD, parallelized as Hogwild [137] are usually used in practice to parallelize computations on multiple machines. This can introduce further noise, however, the same analogy can be made as with pure SGD.

Gradient checking With the advent of modern auto-differentiation modules included in virtually all popular deep learning frameworks, gradient checking is often overlooked. The gradient can be computed either numerically or analytically, the latter method being faster (and now ubiquitously automatized). Gradient checking implies the verification of the numerical gradient against the analytical gradient and is a good idea when custom loss functions are introduced. The numerical gradient computations are based on the centered difference formula $[f(x+h) - f(x-h)]/2h$ where h is usually taken as a very small constant, usually 1e-5.

Momentum The minimum necessary requirements to navigate an error surface is the gradient which specifies direction and velocity which is determined by the learning rate η. Momentum, then corresponds to adding acceleration up to terminal velocity, as if mass γ were added to the vector described by the gradient and learning rate, affecting the potential energy. This can also dampen oscillations by adding a fraction γ of the update vector of the past time step to

the current update vector:

$$\begin{aligned} v_t &= \gamma \cdot v_{t-1} + \eta \cdot \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2.50}$$

Typically momentum is set to a value of $\gamma = 0.9$

Nesterov Momentum [130] is an improvement over the standard momentum. The idea is to look ahead before updating the parameters. In essence, an approximation of the next location in the error space is computed first as illustrated in Figure 2.11:

$$\begin{aligned} v_t &= \gamma \cdot v_{t-1} + \eta \cdot \nabla L(\theta_t - \gamma \cdot v_{t-1}) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2.51}$$

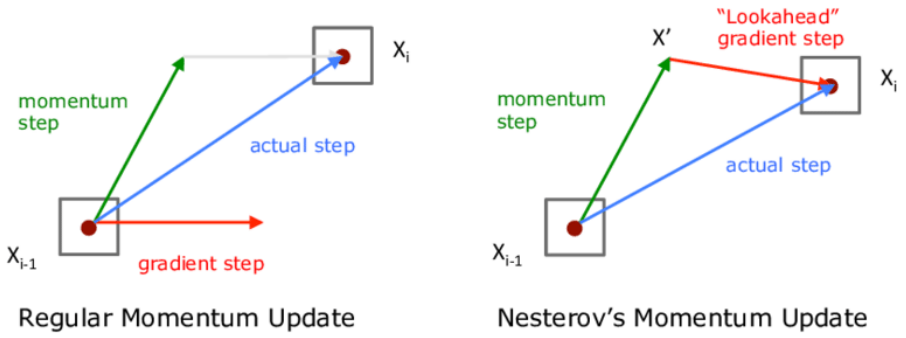


Figure 2.11: Differences between standard and Nesterov momentum.

Adam Momentum adapts the updates to the slope of the error function thus accelerating SGD. However, the same learning rate is used for all parameters. When the features are very different statistically and have different variance in frequency it can be beneficial to update the learning rate for each set of parameters individually, depending on their contribution.

This is what Adaptive Moment Estimation (Adam) [90] does. It stores an exponentially decaying average of past gradients m_t^i and also squared gradients v_t^i for each parameter i. These averages are decayed as follows:

$$\begin{aligned} m_t^i &= \beta_1 \cdot m_{t-1}^i + (1-\beta_1) \cdot \nabla L(\theta_t^i) \\ v_t^i &= \beta_2 \cdot v_{t-1}^i + (1-\beta_2) \cdot [\nabla L(\theta_t^i)]^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \end{aligned} \tag{2.52}$$

In essence, m_t corresponds to the mean and v_t corresponds to the variance of the gradients. The authors propose default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

2.3.2 Second order methods

In the previous section, quite a few tricks where developed for adapting and adjusting the learning rate ∇. In second order methods, there will be no learning rate since it is automatically adapted based on the curvature of the error surface.

Newton's Method is an iterative method used in optimization which does not use any learning rate. The optimization process involves iterating the following update:

$$\theta_{t+1} = \theta_t - H(\theta_t)^{-1} \cdot \nabla L(\theta_t) \tag{2.53}$$

where $H(\theta_t)$ is the Hessian matrix (the square matrix of second-order partial derivatives of the function) and $\nabla L(\theta_t)$ is the gradient. Newton's method will generally require much fewer iterations as compared to first order methods, due to more efficient parameter updates. Since the Hessian describes the local curvature of the loss function, a multiplication with the inverse Hessian results in better calculated steps: larger steps where the loss surface is flatter (there is no risk of going fast since surface doesn't change much) and smaller steps where the curvature is greater (large steps would overshoot the target). The major drawback of the update above is impractical since computing and inverting the Hessian is a very costly process, both computationally and memory wise.

Quasi-Newton aim try to circumvent the memory problem by computing approximations, since the computation of the Hessian is the most costly. Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [131] uses the information in the gradients over time in order to compute an approximation without computing the full Hessian. While this potentially eliminates the memory problem, L-BFGS is still constrained by the fact that the approximation needs to be computed over the entire training data set at once. While it is more challenging to use the mini-batch strategy for L-BFGS, carefully choosing subsets of the training data in mini-batches can be a good starting point. While there have been attempts [148, 125] at adapting these methods for mini-batch and online training for convex problems, generally, using second order methods is not feasible in online computation tasks.

Levenberg-Marquardt The Levenberg-Marquardt [104, 117] algorithm (LM), also known as damped least-squares (DLS) is the fastest known method that can be used to optimize neural networks. It is essentially an interpolation between the first and second order methods. However, it is only useful when the squared loss is used. This algorithm is applied in this thesis in Chapters 3 and 4.

2.4 Related work

2.4.1 Feed forward and convolutional neural networks

Neural networks have been extensively used for univariate time series forecasting. Most legacy reviews vary. [166] compare NNs with classic Box-Jenkins models but do not reach any general conclusions. [15] discusses NNs in the context of forecasting financial markets. [52] provides a comparative study using air line data. [201] also made an extensive study on NNs for time series forecasting, discussing issues (that have mostly been solved in the meantime) finally arriving at an optimistic conclusion. [139] stated that NNs yield the best results when used for monthly and quarterly time series, discontinuous series (missing data) and for large prediction horizons. However, the recent popularity of dilated convolutions (Figure 2.5) make it possible to work with very high resolution time series data and as such the monthly and quarterly data statements no longer hold. Hybrid ARIMA-NN methods have also been developed [200] however these have not been very popular.

Sometimes NNs in the context of time series forecasting have been renamed, although the architecture is almost the same. Time Lagged Neural Networks (TLNNs) [52, 88] are essentially FFNNs applied to time series forecasting where the input data are the time lagged, seasonally selected past observations, just as I demonstrate in Figure 2.1. Also related is perhaps the earliest sequence to sequence model, the Seasonal Artificial Neural Network (SANN) architecture [65], which extends the latter idea by aiming to predict multiple time steps in the future. This transforms the autoregression problem into a multi-task learning problem where each prediction horizon is a different task. [40] provides an overview of deep neural network methods for speech recognition and related applications. However, the connection with Convolutional Neural Networks (CNN) was never clearly made hence simple CNNs used to be the norm.

The foundational work on CNNs applied to images, voice and time series was pioneered by [101] which demonstrate the earliest successful applications. CNNs have also been used for human activity recognition - a multichannel time series forecasting problem [194]. Recurrent CNNs have also been proposed for

[136] scene labelling, also related to time series. The main goal was to increase the size of the receptive field while keeping the model at a reasonable size. No comparison with dilated convolutions was made, however it is unlikely that the latter will not outperform the recurrent CNNs simply due to parsimony.

Beyond time series, there is an abundance of literature on CNNs for computer vision problems. Most of the work was pioneered by [97] on the ImageNet dataset. The ones that are related to time series and worth mentioning are mostly research in the video domain. From these the fast R-CNN [58] is also a MTL application used for object detection that also gives the region of interest (ROI). MobileNets [75] and Xception [33] are also worth mentioning for the depthwise separable convolutions.

Recently, [16] have made an empirical comparison of CNNs with dilated convolutions (most likely inspired by the WaveNet [170]) against RNNs, GRUs and LSTMs on datasets that are commonly used to benchmark RNNs on sequence modelling tasks. The proposed temporal convolutional network (TCN) architecture outperforms the RNNs on all benchmarks and in some cases by a large margin. The research can be considered controversial since it is known that RNNs are difficult to tune properly. Therefore a hard conclusion can not be taken before the results are reproduced. However, it is clear that CNNs are definitely a formidable alternative to RNNs for time series problems, and are most likely to become more popular in the near future since they are easier to train / tune than RNNs.

A further discussion on CNNs is beyond the scope of this thesis since the literature is mostly computer vision related. For a comprehensive discussion on deep learning methods the reader can consult [147, 3]. The latter survey covers the following architectures: CNNs, RNNs including Long Short Term Memory (LSTM) Gated Recurrent Units (GRU), Auto-Encoders (AE), Deep Belief Networks (DBN), Generative Adversarial Networks (GAN), Deep Reinforcement Learning (DRL), exemplifying applications.

2.4.2 Multi-task learning

Multi-task learning (MTL) amounts to sharing parameters (or learned representations in the deep learning context) between related tasks, in order to lower the generalization error. In a nutshell, "Multitask Learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias. It does this by learning tasks in parallel while using a shared representation;" [22].

The idea of MTL can also be expressed biologically. People get better at a certain task by leveraging knowledge learned on related tasks. This is an innate ability. For example, babies learn to recognise faces before they are able to recognize objects since it is more useful to recognize carers before objects or actions.

MTL can also be seen as a data augmentation process. Data from other tasks can aid in filling in the data point gaps which would otherwise cause the models to under or overfit on their own. From another perspective, sharing data between tasks can cause sparse high dimensional features to be selected better.

Applications When it comes to the applications, MTL has been successfully used in many domains such as such as audio [69, 40], vision [58, 146, 109], language [38, 183] and many others. MTL has been rediscovered and referred to in many ways, such as learning to learn, learning with auxiliary tasks, joint learning - to name a few - and is closely related to transfer learning as a form of inductive transfer that causes a model to narrow down the hypothesis space.

General MTL strategies Before discussing MTL in a deep learning context it is best to consider MTL in a classic, well understood setting, namely multivariate regression. In section 2.2 I show that MTL is a generalization of the classic problems of vector autoregression (VAR), multivariate regression (aka vector valued regression) - these can also have a lot of names.

[4] provide an in depth survey of the special case of vector autoregression (VAR) in multi-task learning with multivariate kernels. The survey is broad, well structured and provides multiple perspectives on the learning problem,

from both a regularization theory perspective as well as a Gaussian processes perspective. The latter they show is equivalent to having one generative model per output (task) that expresses correlations as a function of the output index and the input space, using a set of common latent functions (shared representation).

Multiclass classification is in turn a special case of VAR (section 2.2). This is the case since the loss can also be interpreted as optimizing a sum of losses, one for each task. In general, when more than one loss function is being optimized, it is a good idea to think of the problem from a MTL perspective. For example, it is usually the case and a common trick to add fake (auxiliary) classes for a multiclass classification problem in order to improve generalization error.

[141] show that unrelated (i.e. auxiliary) tasks can be exploited in order to increase the generalization error on the main task group. The same input data is used for all tasks and the regularizer encourages a low rank representation and penalizes the inner product between pairs of parameters from the main and auxiliary task groups. The intuition is that this process leverages the prior knowledge of which tasks are related and which are not, thus inducing sparsity in the learned representations. The method is validated empirically on synthetic data as well as two real datasets, one for facial expression with 10 subjects and 7, mutually exclusive emotion classes and one for pain level rating also based on facial expressions. In all cases the proposed method was significantly better, however the datasets were small for current standards.

Another application where an auxiliary task helps training is the work of [9] which jointly learns to model the audio signal and the language features simultaneously. Chapter 5 describes a similar approach for music where the auxiliary task is predicting the midi note using an autoencoder architecture in an autoregressive way. The latter is not used at generation time and is only there to guide the training process. Adding an auxiliary task which is known to enable the model to learn transferable representations has been demonstrated for language modelling [138]. The sequence labelling framework uses an auxiliary training objective, in order to learn to predict the surrounding works for every word.

Although a completely different vein of research, MTL has also been shown

to improve generalization error in decision tree learning [22]. This is a given for multiclass problems, however boosted decision trees [28] can enable data sharing and regularization, demonstrating that decision trees are no exception when it comes to learning tasks jointly.

The Natural language processing (NLP) literature is also a good resource for conceptualizing related MTL problems. The concept of shared representations has been recently adapted to various settings. [183] propose label-embedding attentive models where the words and labels are embedded in the same joint space by measuring the compatibility of word-label pairs to attend document representations.

Also related is domain adaptation where data from a domain is used to transfer the representations to a similar task for which there are no labels. [55] propose an adversarial loss and a gradient reversal layer in order to disentangle representations by adding an additional layer that predicts the domain of the input (either source or target). This forces the model to learn representations that cannot distinguish between domains.

Regularization As exemplified in Equation 2.47 there exist various mixed norm combinations, which are known as block-sparse regularization, while for the ℓ_1/ℓ_2 norm it is known as group lasso [197]. Other mixed norm combinations have been proposed, such as ℓ_1/ℓ_{inf} [199]. Intuitively in practice this is beneficial only if there are major redundancies in the input space. This is why in section 2.2 the input space is repeated in Equation 2.46.

These sparse methods however are not very popular for neural networks since other sparsity inducing strategies such as Dropout [71, 156] or DropConnect [182] are easier to implement and deploy, and MTL regularization obtained using the trace norm as described in section 2.2.

Optimizing the non-convex group lasso can be made convex [8], by penalizing the trace norm of Θ and alternating optimization, as described in Equation 2.39. There also exist various extensions of the group lasso to cases where tasks have an inherent tree structure [89]. Online multi-task learning has also been demonstrated for linear models where the task structure is known [25] or learned from the data [142].

MTL in neural networks As shown in section 2.2 any multiclass classification problem or multivariate regression is an instance of a MTL problem. In addition to regularization there are architectural tricks that can be used in order to enforce parameter sharing. The typical design pattern is to use shared parameters (layers) for the first layers of the NN and then split (independent layers) the last parameters Figure 2.12 - left. This is known as hard parameter sharing [22] and in many ways it is still the most frequent approach for applications where the tasks are closely related. Alternatively, sharing can be enforced by linking layers from identical architectures, one per task. This is known as soft parameter sharing.

Another way to classify neural network MTL approaches is based on the type of outputs. In homogeneous MTL, each task corresponds to a single output such as in multiclass classification. A typical example is the MNIST digit recognition [98]. In contrast, in heterogeneous MTL every task has it's own set of outputs as it has been demonstrated for facial landmark detection [203]. Another example is detecting the type of object (task one) and it's placement in an image (task two) [58] or multiple such related tasks such as street classification, vehicle detection and road segmentation [167].

In the most common way of sharing parameters in neural networks - hard parameter sharing - the first input layers are shared and the last few layers of the network is kept task specific [22]. Keeping the last layers separate is not strictly necessary, as it is the case with multiclass classification. This is the main clue to why hard parameter sharing lowers the chance of overfitting. The gradients from each task help find common representations that are more robust. The more tasks, the better features are learned in the shared layers.

In more complex circumstances, if vector outputs would have to be produced for each class such as in object detection for multiple classes, then it is preferred to have separate last layers per task. However, an even better strategy as it turns out is to separate the prediction of the class label on a discrete loss output layer and the coordinate vector indicating the object location as a regression output layer [58]. The authors in the latter work show that MTL training improves the results.

Soft parameter sharing implies learning how to share parameters as well,

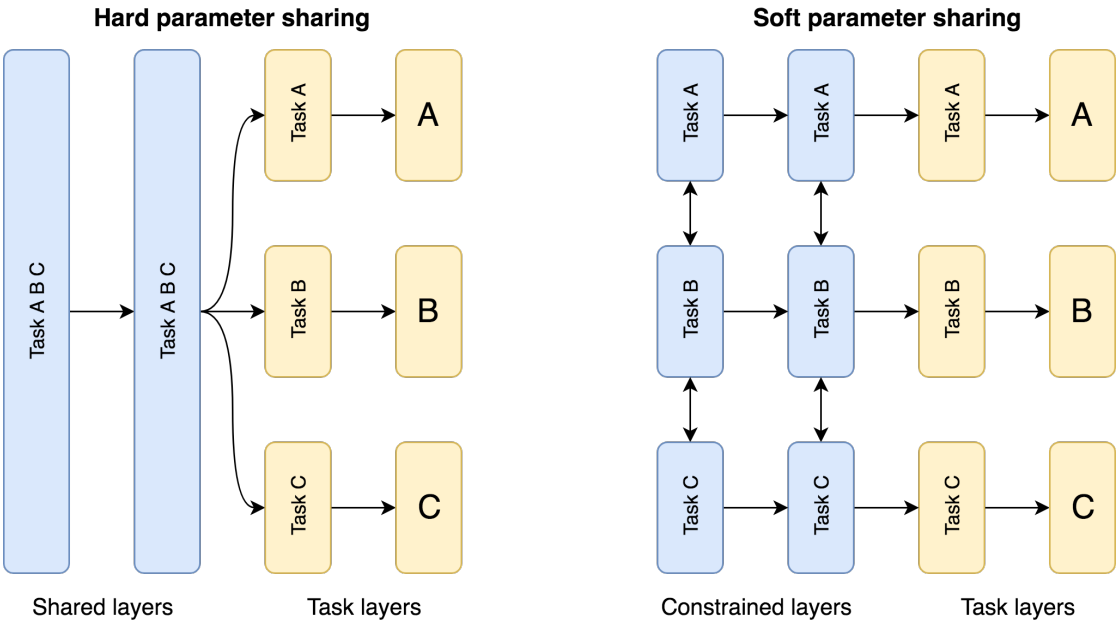


Figure 2.12: MTL parameter sharing strategies for neural networks.

and applies the regularization techniques form linear and kernelized models [4] while the MTL architecture can be seen as having one model per task (Figure 2.12 right). Different regularization strategies exist and choosing an explicit strategy depends on the specific problem. Recent work however has shown that designing a specific architecture can improve accuracy over an automatic approach to learning how to share.

Implicit architectures A typical example of soft parameter sharing is the work of [123] where two separate architectures are used. Special architectural units that learn linear combinations of the output of previous layers are used in order to enable the models to share parameters and leverage knowledge.

An interesting approach [112] to learning structure as well as the parameters to be shared starts with a thin shallow network that is gradually increased during training in a greedy way. A criterion is used which encourages the grouping of similar tasks. This method is more reminiscent of growing decision trees, since the final architecture resembles a tree of layers. Having a narrow layers at the start and a wide ones towards the outputs is somewhat unintuitive since the

most generic learned representations are also more numerous. In addition, one branch is allotted per task, which does not considerably encourage learning of complex representations and sharing topologies between tasks.

In a similar attempt to automate the architecture design process, the work of [195] reduces the design choices for MTL architectures by proposing a matrix factorisation approach in order to flexibly share knowledge in fully connected and convolutional layers. The parameters are shared across tasks at every layer of the network. The method shows improvements over both homogeneous and heterogeneous datasets over single task approaches, as demonstrated for two vision problems. The latter work is based on and generalizes the work of [98] where a structure constraint is placed on the parameters factorising the shared parameters and the task-specific parameters. While the approach in [195] is generic and flexible, crafting special architectures usually leads to better results.

Explicit architectures As previously mentioned there is usually universal solution and specific applications are better off with unique solutions.

Starting from what they call the uncertainty of each task [87] and with a fixed structure of sharing, the relative weight of each task is adjusted using a loss function that penalizes task uncertainty. The approach is applied to instance and semantic segmentation as well as per-pixel depth regression.

Recently, encoder-decoder architectures have been shown to be successful in deep MTL learning. Such an approach is taken by [167] where a single decoder is used to extract rich shared features and three separate decoders are used for classification, detection and segmentation.

The work of [126] also uses a WaveNet [170] encoder to learn a shared generic representation of music and several decoders are used in order to translate the music from one instrument timbre to another via an domain adaptation approach.

Similarly, [82] use convolutional, attentional [176] and mixtures of experts architectural blocks, sharing as many parameters as possible, into a single deep learning model which jointly learns several large-scale tasks from multiple domains. Each task has it's own encoder and the decoding is done via a mixture of experts decoder [151]. These blocks are combined in a similar manner to

WaveNet [170]. The architectural mechanisms devised for different domains are combined (NLP, computer vision) and the authors find that adding these architectural blocks does not hurt performance, even on tasks they were not designed for.

In conclusion, it appears that hard parameter sharing along with combining architectural blocks form different domains in an autoencoder-like fashion is the main recent paradigm, with state of the art results. This is reminiscent of latent variable models (possibly recurrent) which are discussed next.

2.4.3 Deep generative models

There are essentially three main types of generative models: latent variable models, adversarial models and autoregressive models. I will first briefly discuss the first two after which I will discuss autoregressive generative models with a focus on sequence modelling and give comparisons with other alternatives to generating sequences. A review of deep generative models from the perspective of graphical models is also provided in [134]. The taxonomy is based on the learning algorithm and architecture. Here I adopt a different taxonomy in order to highlight the state of the art methods and a brief history.

Latent variable models One of the earliest latent variable model was the Deep Belief Network (DBN) [143]. DBNs are composed of several stacked Restricted Boltzmann Machines (RBM) [70]. Each RBM has two layers, namely the input layer and a hidden layer (the latent space) which captures high-order correlations of the layer before. These used to be popular since there were no vanishing or exploding gradient problems. RBMs were also adapted for temporal problems (RTRBM) [161]. Since each RBM needed to be trained one after the other, the training procedure was more inconvenient. Hence, DBNs most likely dropped out of favour because of neural networks with ReLU units which were trained in one shot via backpropagation.

Plain Autoencoders (AE) [96] or Denoising Autoencoders (DAE) [177] are also a fundamental generative latent variable models. These, as well as DBNs were used in order to pre-train large networks consisting of many layers [178].

This is no longer common since the introduction of the ReLU activation [97] and especially residual connections [67]. Convolutional Autoencoders have also been developed [118]. The applications are also vast, especially in unsupervised learning such as speech denoising [111] visual tracking [184] and learning word representations [6].

The main problem with AEs and DAEs was that there was no structured and deterministic way of sampling from the latent space. This is why Variational Autoencoders (VAE) were introduced [92]. It did, however, take a considerable amount of time until the potential of the method was understood. A tutorial is provided in [46]. In recent work [31], these have also been combined with autoregressive generative models in order to remove detail via lossy compression (latent variable models are in essence performing compression).

[68] proposed a variation of the VAE by introducing an adjustable hyperparameter that balances latent channel capacity and independence constraints with reconstruction accuracy. The method is more stable to train and outperforms InfoGAN [30] (discussed later) without making assumptions about the data distribution.

The most recent work that makes it clear that the architectural choices are not necessarily linked with the training procedure is [149]. The authors explore the effect of architectural choices (CNNs, RNNs and FFNNs) on learning VAEs for text generation and propose a hybrid CNN-RNN architecture. Furthermore, the VAE framework works with virtually any type of data whether discrete or continuous, sequential or static in unsupervised, semi-supervised and supervised modes which is why the VAEs are very powerful.

More complicated architectures in the VAE framework exist. For example the Deep Recurrent Attentive Writer (DRAW) [61] mimics the foveation of the human eye while the Attend Infer Repeat [51] architecture is able to attend to scene elements and processes them one at a time thus learning to identify multiple objects fully unsupervised.

Adversarially trained models GANs [59] more than any other method piqued the interest in generative models, perhaps also because the confusion with adversarial learning. The idea is to introduce an auxiliary model that acts like a

discriminator. The latter is optimized simultaneously with the generator network, hence the adversarial naming.

InfoGAN [30] extended GANs to learn interpretable representations. This was an improvement since in GANs the layout and organization of the latent coding space is underspecified. Hence, InfoGANs minimize the mutual information between small subsets of the latent parameters learned and the observations, thus imposing additional structure.

SeqGANs [196] have extended GANs to sequences by modelling the data generator as a stochastic policy in reinforcement learning (RL). These are of course more complicated to train than autoregressive convolutional models, however should be faster at generation time. No direct comparison with WaveNets [170] was made. It is difficult to compare the performance of generative models [168].

WassersteinGANs [10, 63] minimize the Wasserstein distance between the data distribution and the generator distribution. The most recent improvement was the [202] SAGAN (Self-Attention Generative Adversarial Network) which aims to balance the long term dependencies due to the large receptive field and efficiency via a self-attention layer in both the discriminator and the generator.

Autoregressive models These are also latent variable models with the addition of autoregression. Obviously RNNs are the norm here, and the oldest such model is the LSTM [72]. Perhaps the fist modern deep learning generative model was the Neural Autoregressive Distribution Estimator (NADE) [100]. Perhaps the best way to explain the latter is in relation to the DAE. Masking noise is used in both models, however NADE is trained based on the average reconstruction of only the inputs that are missing.

Later, these were upgraded to the Masked Autoencoder for Distribution Estimation (MADE) [57]. The main difference is that the masks used are designed such that the outputs are autoregressive towards the outputs - the target outputs are shifted full versions of the inputs, and thus the generated outputs can be interpreted as conditional probabilities. In other words, each input dimension is reconstructed only from the dimensions preceding it in the ordering (i.e. causal).

PixelRNN [133] were the first autoregressive generative models that produced good results on the ImageNet dataset by sequentially predicting the pixels in an image along the two spatial dimensions, similarly to a character level RNN. The advantage is that it can fill in occlusions from unseen images and of course interpolate between several images.

PixelCNN [171] is the adapted and improved version of the PixelRNN architecture, greatly reducing the computational cost. The architecture returns explicit probability densities, making the application to compression and probabilistic planning and exploration straightforward.

The latter were also later improved by the PixelCNN++ [145] where the discretized logistic mixture likelihood [93] is used, as opposed to a 256-way softmax. This further increases performance since the softmax does not preserve the quantized sequence order (i.e. the model does not know that a value of 128 is close to a value of 127 or 129). Usually this is an undesirable property for classification, however when order is important it is beneficial.

WaveNets [170] where the natural extension of the PixelCNN to the raw audio domain, essentially advancing the Text To Speech (TTS) state of the art to realistic human voices. Since then there have been several extensions to music and other domains [16]. Even though these models generate audio that sound the same, the waveforms were different and the fidelity varied.

The VQ-VAE (Vector Quantised Variational AutoEncoder) [172] is also worth mentioning since it facilitates better interpolation within the layers of the latent codes, similarly to the VAE framework. The main difference is that the encoder operates on discrete codes as opposed to the continuous case of VAEs. The framework is applied to WaveNets demonstrating speaker conversion and other tasks.

SynthNet (chapter 5) improves upon WaveNets by generating high fidelity music which is virtually undistinguishable from the original samples SynthNet is trained in an end-to-end fashion. This is also made possible by the auxiliary NADE-like midi prediction task. It also trains much faster since the parsimonious approach leads to increased computational efficiency over the WaveNet predecessor.

Summary of methods All of these methods are powerful since they allow the controlled generation of examples that interpolate between the points in the training data. This is also known as learning to disentangle, or disentangling manifolds. Interpolation is best exemplified for VAEs with images. In Figure 2.13 below the two dimensional space of the AE and VAE latent codes are compared for the MNIST dataset:

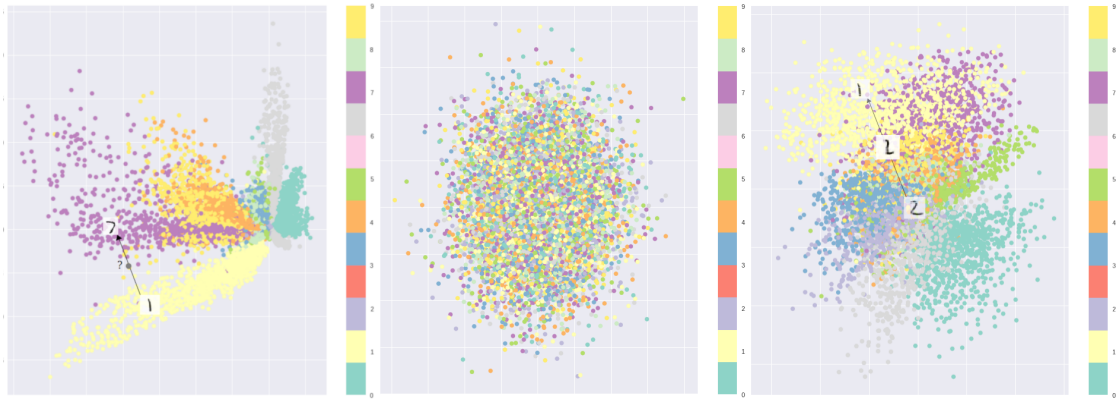


Figure 2.13: Left: AE optimized only for reconstruction loss. Middle: VAE with pure KL loss results in a latent space where encodings are placed near the centre without any similarity information kept. Right: VAE with reconstruction and KL loss results in encodings near the centre that are now clustered according to similarity.

However, all the state of the art approaches (VAEs, GANs, WaveNet / PixelCNN) have both advantages and disadvantages. VAEs allow for both learning and efficient Bayesian inference in any type of NN architecture (i.e. graphical models with latent variables) however, for images the generated samples are more blurry than other methods. In contrast, GANs generate the sharpest images however they are much more difficult to train and condition in order to generate the desired parametrized samples (i.e. less control on what is being generated).

The best plausibility of the generated data (i.e. fake data) is given by the conditional autoregressive models (PixelRNNs) - these are also relatively easier to train compared to the other two - however the major drawback is that generation is slow (see Figure 2.8) and there isn't any solid understanding of the learned representations. This thesis thesis makes some progress towards explaining the learned representations for the latter models, as studied for music.

Furthermore, I also considerably reduce the training time and therefore generation marginally. Finally, the generated data was previously from a similar distribution but different while now the generated data is nearly identical to the ground truth which allows a more direct comparison.

Invertible generative models Recently, the GLOW (Generative Flow with Invertible 1x1 Convolutions) model was proposed [91] which can generate high resolution realistic images, can be sampled from efficiently and with a meaningful mapping of the discovered features that can be controlled at generation time. This model was based on the NICE [44] and the [45] methods. Essentially, the improvement over autoregressive generative models is the ability to parallelize the generation process.

In [60] a class of generative models is introduced, that does the mapping from simple to complex distributions via invertible neural networks. The continuous-time invertible generative model has an unbiased density estimation and sampling is done in one pass, which also enables fast generation.

Chapter 3

Traffic forecasting in complex urban networks: Leveraging Big Data and machine learning

This chapter is based on the following peer-reviewed publication:

F. Schimbinschi, V.X. Nguyen, J. Bailey, C. Leckie, H. Vu, R. Kotagiri, "*Traffic forecasting in complex urban networks: Leveraging big data and machine learning*" in **International conference on Big Data**, (IEEE BigData) pp. 1019-1024, IEEE, 2015.

3.1 Introduction

This chapter studies the data and pre-processing aspects involved in sequence modeling applied to peak traffic forecasting. Simple regularized convolutional models are benchmarked against other established classification methods for efficient and accurate forecasting. This chapter shows that the availability of big data helps towards prediction accuracy. Furthermore, the spatial dimension has more influence on the results than the temporal one. Finally, careful choice of thresholding parameters is crucial for accurate classification.

The availability of detailed data streams on road networks offers great promise

for intelligent transport in the context of smart cities. Applications include personalized copilots with real time route suggestions based on user preferences and traffic conditions, economical parking metering, agile car pooling services and self driving cars. At the core of these systems, a proactive and accurate, network-wide, real-time traffic prediction system is paramount.

Current systems are largely reactive since much of the existing work has been performed on simple freeway datasets that do not entirely capture the complex spatiotemporal characteristics of a city's traffic. Here, an intricate network dataset is studied in the context of traffic forecasting in complex urban networks.

The main contributions are:

- The investigation of a novel big traffic data set
- An exploratory analysis of the data
- Suggestions on how to tackle pre-processing for a forecasting problem
- A benchmark of state of the art machine learning algorithms
- In the process, the following questions that are relevant for any multi-task learning problem are asked:
  - Is a larger temporal context beneficial?
  - If so, is it because it captures the large variance between weekends and weekdays?
  - Does augmenting missing data with contextual average trends increase accuracy?
  - Is recent data more useful on its own?
  - Does big data help even if old data is used?
  - Are there advantages to modeling the forecasting problem as a multi-task learning problem?

The experiments reveal that: i) modeling prediction as a multi-task learning problem is beneficial, ii) the spatio-temporal representation is one of the central

issues, iii) predicting only on weekdays is easier and separate predictors can be deployed separately for weekends or each day of the week, iv) adjusting the receptive field size and proximity data increases performance, v) for classification problems, the quantization of real-valued data is a central issue and should either be avoided or thresholding should be set dynamically.

3.2 Dataset

The VicRoads dataset was collected in the City of Melbourne over six years. A special feature of this dataset is its volume and variety, covering the Central Business District (CBD) and suburban areas, including freeways as depicted in the figure below. A quantitative comparison with existing datasets is given in Table 3.1. To the best of our knowledge, it is the first dataset of its kind studied by the community. Vehicle volume count data is recorded using loop detectors at a frequency of 1 minute for the raw data set, which measures about 700 GB per year. VicRoads recently released[1] all their fine-grained traffic volume data. Here a subset is used.

The data stream is aggregated into 96 bins for each day, taken at 15 minute intervals for each sensor. Measurements are aggregated for all lanes in the same direction, aggregating into road segment level statistics. These tensors thus contain traffic information for 2033 days $\times$ 1084 sensors $\times$ 96 time points per day.

3.3 Exploratory data analysis

A random subset of days was gathered for one sensor and the 96 dimensional traffic volume vectors projected onto a 2D space using a randomly generated orthonormal matrix. Random embeddings onto lower dimensions preserve Euclidean geometry, thus three clusters are visible in Figure 3.2.

Examining several samples from each cluster suggests the largest variance between days corresponds to whether the day is a working day or part of the weekend. The same can be observed in Figure 3.4 where the daily average traf-

[1] `https://vicroads-public.sharepoint.com/InformationAccess/SitePages/Home.aspx`

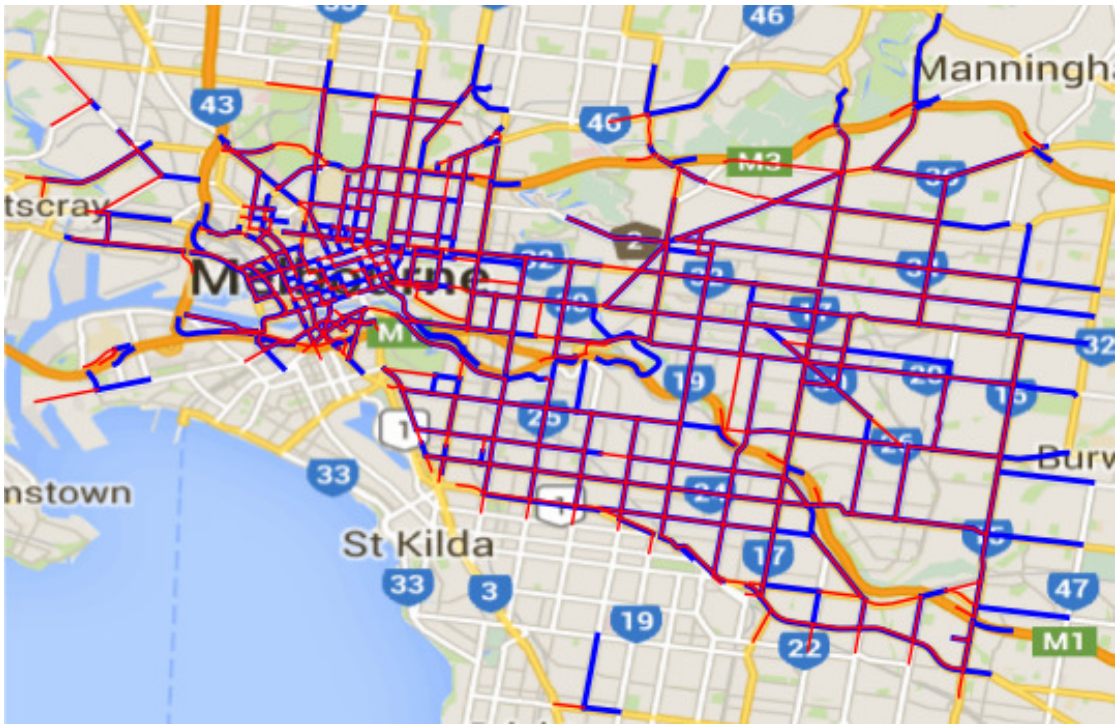


Figure 3.1: Melbourne roads with available traffic data are highlighted. Each physical road typically has 2 traffic directions, coloured red and blue. Latitude and longitude coordinates for each sensor are also included.

fic volume was computed for each day of the week, confirming the weekday – weekend assumption. The high density cluster (diamond) in Figure 3.2 corresponds to days where the volume is zero. This may be due to sensor failure, maintenance operations or human processing errors. The number of days having such events were counted for each sensor and sorted:

In total there are 128,014 zero-valued days out of 2,203,772 (5.8%). The distribution differs across sensors. Approximately 92% of sensors have less than 10% missing data, while the rest of 8% can reach up to 56%. The cumulated averages per each day of the week show that for working days, the largest variance between sensors is the difference between noon and evening rush hour peaks. For some sensors the traffic peaks are higher at noon and lower in the evening while for others, the peaks might have the same amplitude. It is interesting to observe in Figure 3.4 that for Fridays and Saturdays there is a another volume peak at approximately 23:00. However, this pattern is not observable for all sensors.

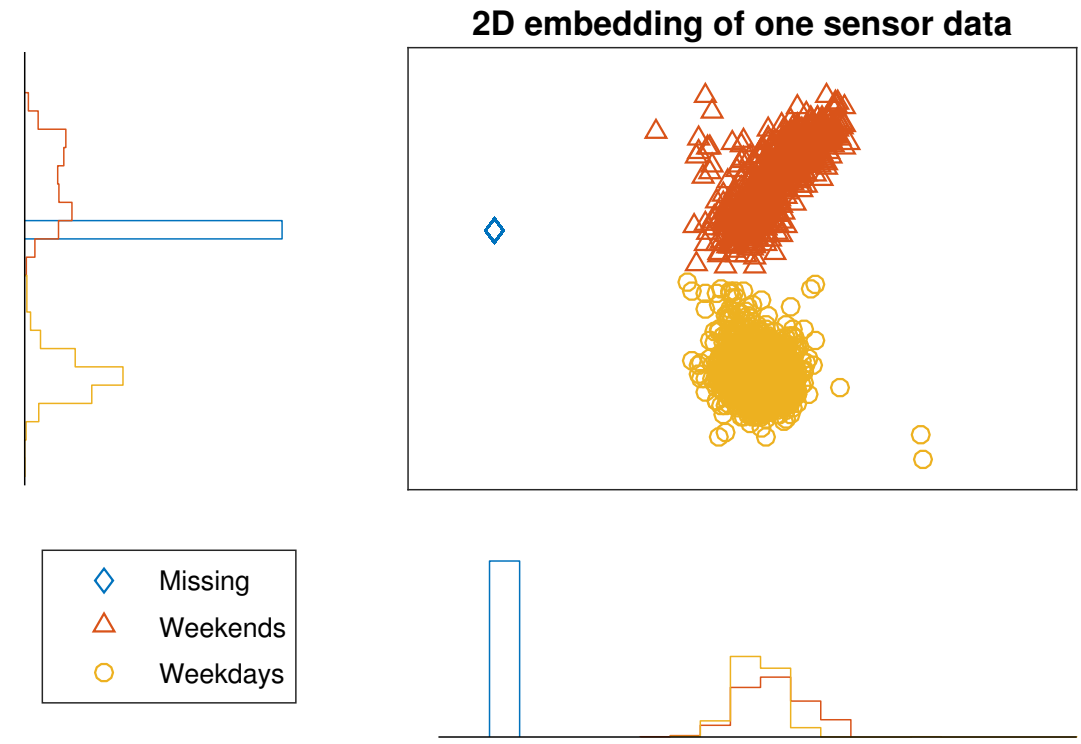


Figure 3.2: 2D embedding of a subset of data for one sensor. Large clusters are weekdays or weekends. The high density diamond cluster corresponds to days where the traffic volume is zero for an entire day (missing data).

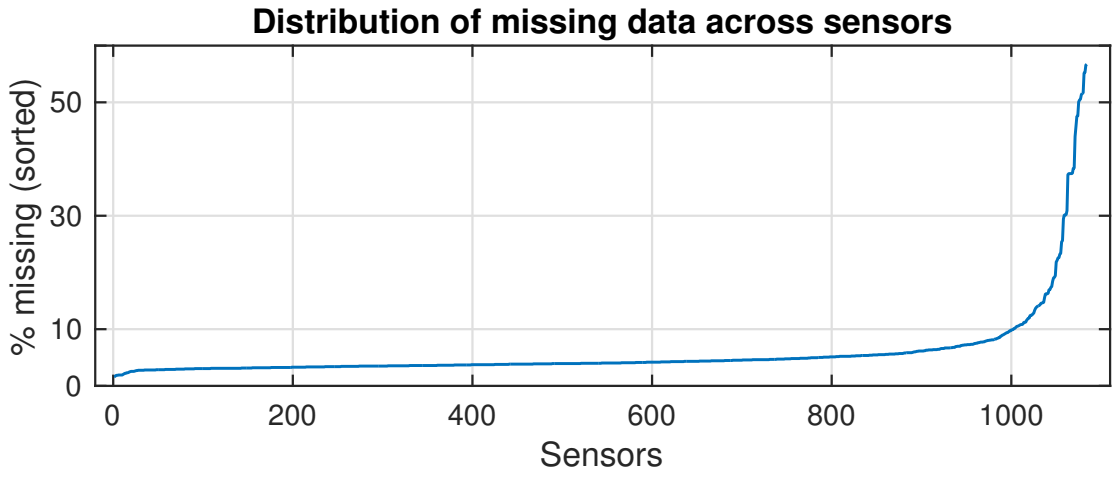


Figure 3.3: 92% of sensors have less than 10% missing data, while the rest can reach up to 56%. There are only 10 sensors without missing data.

Another source of variance between sensors is a slight time shift between peaks (e.g. morning traffic peaks between 07:30 and 08:30). These shifts are likely to be a function of the sensor's proximity to the CBD. During weekends, the traffic pattern is not consistent across sensors, though typically there is a noon peak (12:00 – 13:00).

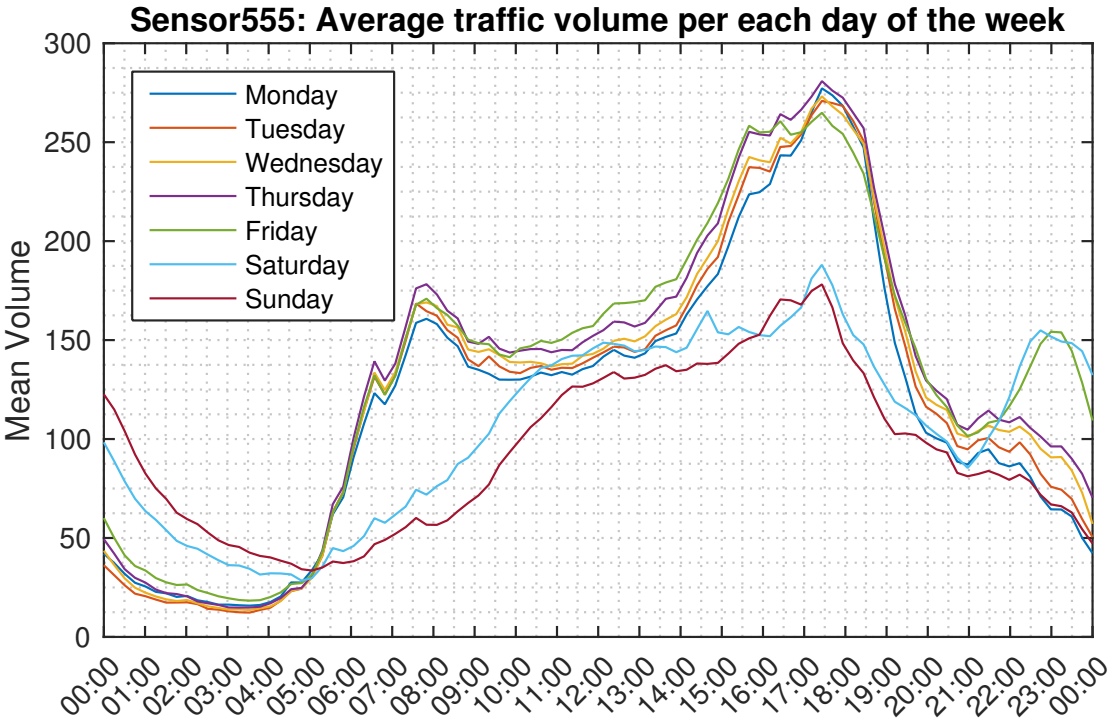


Figure 3.4: Average traffic over 6 years accumulated per day of the week. An outbound road. Evening peak is higher, when commuters depart.

Sporadically, weekend evening peaks are similar to a typical weekday peak. Another interesting observation is that even though some sensors have the same peak traffic profile across working days, the actual *volume* can differ across days. Each day could thus be modelled individually. A similar analysis was performed in [34], where the daily characteristics show similar patterns. The authors in the latter conclude that seasonal variance is mainly governed by school holidays.

3.4 Problem setting and related work

For prediction, the dataset was reshaped into a single continuous series per sensor. Each column S is a sensor, scaled to $\mathcal{D} \in [0, 1)$. This yielded a dataset $\mathcal{D} \in \mathbb{R}_{+}^{T \times S}$ where $S = 1,084$ sensors / tasks and $T = 195,168$ time points. The first 70% time frames were used for training while the last 30% was kept unaltered for testing in all experiments.

3.4.1 Spatio-temporal considerations

The term *n-step-ahead* refers to the number of points into the future for which predictions are made. *Prediction horizon* refers to the difference between the current time and the start of the prediction time and can be one-step-ahead or n-step-ahead. Here, only one-step-ahead predictions in the immediate step into the future are considered. It is trivial to adapt the forecasting problem to a larger prediction horizon.

Simultaneous network-wide predictions can be modelled either as multiple individual learners as part of a multi-task learning problem that makes predictions on all measurement points simultaneously. Initially, each sensor is modelled individually, only with it's own data. For one-step-ahead prediction, the dimension of the response variable (target) $y_s(t) \subset \mathcal{D}^s(t)$ where $y_s \in \{0, 1\}$ is

Table 3.1: Comparison of the VicRoads dataset with ones from literature.

Dataset	# Sensors	Timespan	Granularity	Total timepoints
VicRoads	1084	6 years	15 Min	211,562,112
[85]	837	3 months	5 Min	20,248,704
[120]	502	1 week	5 Min	1,012,032
[135]	52	31 days	5 Min	464,256
[122]	50	16 days	5 Min	230,400
[32]	22	24 days	5 Min	152,064
[191]	4	28 days	5 Min	32,256
[41]	4	10 hours	20 Min	>5,000
[102]	12	6 days?	5 Min	1,600
[185]	4	??	1 Min	--

always one $|y_s| = 1$.

The sliding window (receptive field) is moved forward one step at a time through the training set for all sensors simultaneously. The training data $x_s(t) \subset \mathcal{D}^s(t-1-\Delta, t-1)$ with $x_s \in [0,1)$ has a length of $|x_s| = \Delta$ observations and is sampled for a particular sensor and a specific time frame, while the window is moved. f_s is the decision boundary, ε_s is the irreducible error and λ_s is the regularization term for sensor s. Finding the optimal decision boundary $f_s \in \mathcal{H}_s$ for peak traffic forecasting can be modelled as a general least squares problem:

$$\arg\min_{f_s \in \mathcal{H}} \{ \|f_s(x_s + \varepsilon_s) - y_s\|_2^2 + \lambda \|f_s\|_2^2 \} \tag{3.1}$$

Thus, there are S such equations that are solved simultaneously although *independently* during the training phase. In subsection 3.6.2 data is additionally shared between predictors. In the case of linear models, solving the equations independently is equivalent to solving them as a full system and any multi-task problem is a generalization of vector-valued learning [17].

3.4.2 Related research and datasets

Predominant methods in the literature are Autoregressive Integrated Moving Average models (ARIMA), Kalman filters, spectral methods and neural networks. A study [35] on short term traffic forecasting suggests that compared to neural networks, the other algorithms are less robust when congestion increases. The work in [18] suggests that this might be due to the smoothing of input data, which obscures the spatio-temporal correlations. In [7], the authors conclude that Big Data is paramount for increased performance.

ARIMA [12] are parametric models commonly used in time series prediction. SARIMA models are used to cope with seasonal effects. VARIMA models generalize univariate to multivariate and capture *linear* correlations among multiple time series. A VARIMA inspired [120] makes predictions as a function of both location and time of day. They report an average accuracy of 91.15 over a network of 500 sensors.

A study on autocorrelation on spatio-temporal data [32] concludes that ARIMA

based models assume a globally stationary space-time autocorrelation structure and are thus incapable of capturing complex dynamics. Another ARIMA inspired algorithm [85] uses a parametric, space-time autoregressive threshold algorithm for forecasting velocity. The equations are independent and incorporate the MA (moving average) and a neighbourhood component that adds information from sensors in close proximity. Lasso [205] is used for simultaneous prediction and regularization. The authors motivate their approach as a means of coping with computational intractability in the case where the number of sensors is larger than 300. In the next sections I show that it is possible to tractably make accurate network-wide forecasts on 1084 sensors simultaneously.

Particle filter methods have been used for traffic state estimation on freeways [185, 191], in combination with other methods such as discrete wavelet transforms. Similar to [41], such datasets are quite different to ours: the focus is on high resolution time-series on short intervals. Freeway data is less complex and furthermore these algorithms are challenging to fine-tune [185]. As pointed out in [36] such methods are largely reactive. Moreover, particle filters are difficult to scale to large nonlinear road networks. A nonparametric (kNN) multivariate regression technique is evaluated in [36] for one-step-ahead forecasting. The term multivariate refers to the modeling of three types of measurements, namely velocity, volume and flow. The authors show that using data from multiple types of measurements increases performance.

Neural networks have been used extensively for short-term real-time traffic forecasting [35, 152, 18, 49, 41, 102, 135] where the focus is to predict on larger prediction horizons. However, the employed datasets are far too simple. In [135] a neural network is used for simultaneous forecasting at multiple points along a commuter's route (the route is set and prediction is done before the travelling starts), with an error averaging to 5 mph for a 30 minute route. Multiple univariate neural networks are used in [102] for prediction. Data from the past week, neighbouring traffic and the day of the week is added as input in order to further improve performance. Recurrent neural networks have demonstrated better forecasting performance [41] at larger prediction horizons compared to feed-forward networks. Hybrid ARIMA and neural networks [200] have also

been applied successfully.

3.5 Peak traffic volume prediction

Peak traffic prediction is modelled as binomial classification. A threshold was set $\omega = 0.85$ for each sensor and high volume $y_s(t) > \omega = 1$ was labelled as positive examples. This procedure resulted in an imbalanced set, since peak traffic is less frequent. Accuracy was used as a performance measure. It is intuitive to expect an accuracy of 85% as a lower bound. Several baselines were evaluated: daily average traffic patterns (Figure 3.4); means from the previous week and ARMA models. The highest accuracy 89.24 was recorded with the cumulated daily average method.

Feed forward and convolutional models A logistic regression model is equivalent to a neural network with no hidden layers. Any dense feed-forward network can be interpreted as a convolutional neural network and vice-versa as explained in subsection 2.1.3. Even the Logistic regression (LogReg) model is in essence a CNN with no hidden layer.

3.5.1 Initial algorithm selection

A subset of 15% of the sensors was selected randomly and eight algorithms were evaluated. The results are displayed in Table 3.2 along with average running times in seconds. There were no efforts made towards fine-tuning.

The table above is sorted in descending order on the third column. With $\Delta = 1$ the best accuracies are recorded for the same algorithms that do better with a larger receptive field, suggesting that these algorithms are more appropriate for the current task. Table 3.2 suggests which algorithms to eliminate from further consideration. There is only a marginal improvement for the first four algorithms, while the last three show a sudden decrease in performance, some even falling under the baseline. Increasing window size Δ should not dramatically decrease accuracy. A failure to effectively use the information gathered using a larger time sample suggests that the algorithm is less suitable for

detecting, learning and predicting complex events.

3.5.2 The effect of increasing window size

Since some algorithms in Table 3.2 are slower with the same or worse accuracy, in the follow up experiments only logistic regression, causal Convolutional Neural Networks (CNNs) and classification trees are considered. Results are shown in Figure 3.5 where the window size is increased even further with $\Delta \in \{1, 5, 10, 20\}$.

The accuracy increases linearly for $\Delta > 1$. The behaviour for logistic regression is similar to the CNN. However, the latter always outperforms the simpler logistic. This implies that there must be an intrinsic lower dimensional space where the classes are better separable. The maximum accuracy for the CNN is 93.13 while for the linear algorithm it is 92.83, with $\Delta = 20$ equivalent to looking back 5 hours. For a $\Delta = 10$, the accuracies are 92.99 for the CNN and 92.70 for logistic regression. Elastic Net and Lasso [205] (L_1, $L2$ and $L1 + L2$) were succinctly evaluated for both a linear and quadratic combination of time points. Regularization can decrease variance at the expense of increasing bias. The contribution of each time point in either form is almost equal and thus regularization is not useful in the original space.

Table 3.2: Network wide classification accuracy and average running time on a random subset (15%) of sensors. One independent predictor per sensor.

Algorithm	$\Delta = 1$	$\Delta = 5$	Seconds
Baseline	89.34		N/A
LogReg	92.54	**92.95**	2.5
CNN	92.49	**92.86**	7.2
RUSBoost	90.19	***92.21***	*430.9*
LDA	91.99	***92.00***	0.1
Tree	***92.47***	*90.66*	*0.8*
SvmRBF	91.90	85.34	140.7
NB	91.61	84.00	13.4
kNN	85.06	81.18	42.2

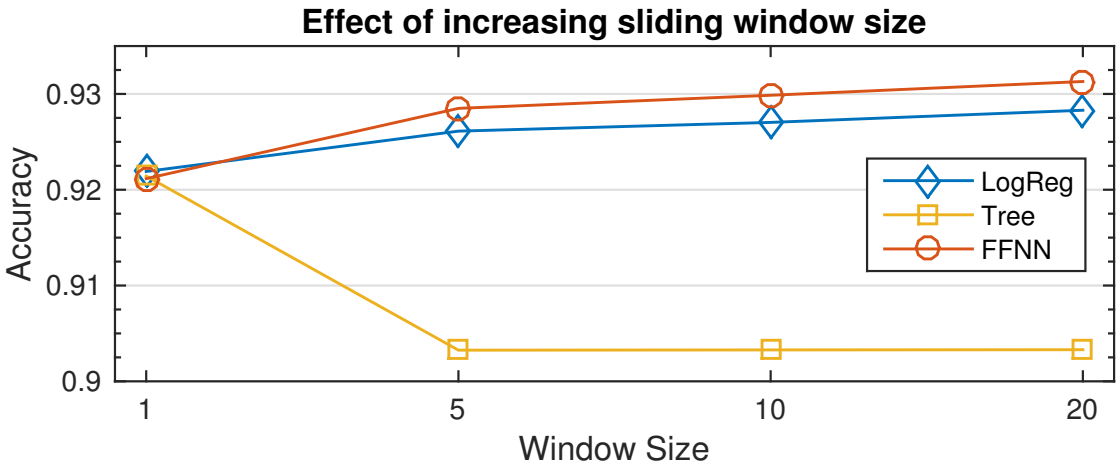


Figure 3.5: Increasing window size w results in better accuracy. Results for $\Delta = 20$: CNN 93.13, logistic regression 92.83.

3.5.3 Exclusive Monday to Friday traffic prediction

From previous experiments it is clear that using more time steps from the past provides a more robust temporal context and thus results in better accuracy. Here, I ask whether this also holds for simpler periodic data, by considering prediction only on working days. This experiment is a follow up on the observed clusters in Figure 3.2. It is possible that a larger window size captures more accurately the difference between work days and weekends, hence the better predictions. Towards evaluating this hypothesis, the weekends were removed from both the training and the testing set and the previous experiment was repeated. These results are not directly comparable to those in Figure 3.5. However, the majority of the literature is focused on Mon–Fri data, captured from less complex traffic networks.

The above figure shows that the hypothesis was false and indeed increasing the window size still has a great impact on prediction accuracy. Despite the fact that the greatest variance is observed between workdays and weekends (see Figure 3.2), simply capturing more temporal context is still beneficial, regardless if the largest source of variance is not present. For all following experiments the window size is set to 10, unless otherwise stated. The choice was made not to take a larger window size for computational reasons.

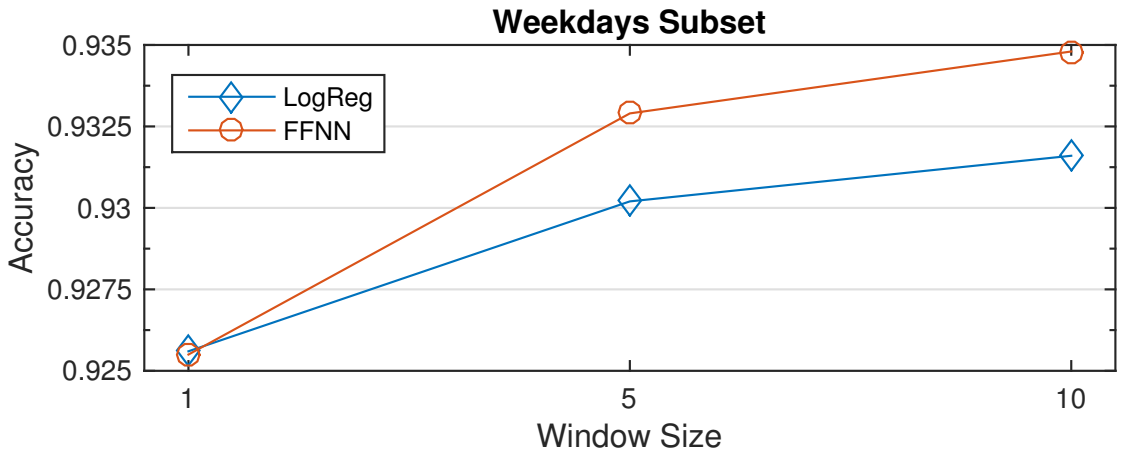


Figure 3.6: Weekends are removed. Larger window size Δ still results in better performance. Top accuracy (93.48, $w = 10$, CNN) is better than if weekends are included (92.99).

3.5.4 Augmenting missing data with context average trends

Missing data is one of the frequent problems of big data applications. This is also a significant characteristic of the current data set. In order to observe the impact of missing data on prediction accuracy, the zero values were replaced with the corresponding hourly sensor trend, cumulated per each day of the week. The results for logistic regression and the CNN are presented in Table 3.3, along with the difference in accuracy Δ from the previous experiment (see Figure 3.5). The augmented dataset is denoted by $\mathcal{D}_\mu$.

Table 3.3: Adding mean trend values for missing data increases accuracy.

	Logistic Regression			FF Neural Network		
	$\mathcal{D}_\mu$	$\mathcal{D}$	Δ	$\mathcal{D}_\mu$	$\mathcal{D}$	Δ
$w = 1$	92.26	92.19	0.07	92.31	92.12	0.19
$w = 5$	92.68	92.61	0.07	92.91	92.85	0.06
$w = 10$	92.78	92.70	0.08	93.05	92.99	0.06

3.6 Big Data versus Small Data

Collection of Big Data is essential, as it is not possible to know what questions will be asked in the future. What does Big Data mean for the current setting? Characteristic to our dataset, there are two dimensions to consider when addressing this question, namely time and space.

How much data is needed in order to generalize well? As one might guess, the traffic patterns are quite cyclical. Towards answering these questions, I repeat the previous experiments and: i) use less temporal data; ii) share data between classifiers, based on sensor proximity.

3.6.1 Leveraging the temporal dimension of big data

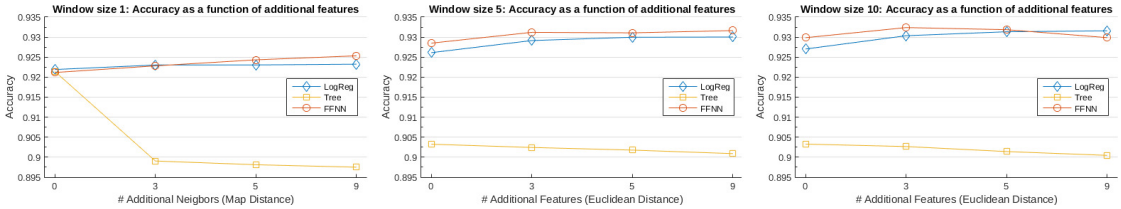


Figure 3.7: Additional data from 3, 5 and 9 closest sensors is added to each classifier. Best result thus far: 93.24% CNN $w = 10$ $k = 3$.

In this section I examine the implications of using less data from the temporal dimension. Table 3.4 shows accuracy as a function of the size of the training set for the logistic regression algorithm and the neural network. The first 10% (old data), last 10% and 25% (recent data) of the training set is selected for training, while the test set is kept the same.

Roughly, 100% of the training data accounts for traffic volume recorded over approximately 4 years, while for the validation set, it accounts for approximately 2 years. Then, 10% of the training data amounts to half a year, while 25% from the entire training set corresponds to data for one year.

Using less data results in a decrease in performance and the drop is more abrupt as data become increasingly outdated. If data only from the first year is used for training (Table 3.4 column 2), the accuracy decreases almost to the

Table 3.4: Big Data is relevant on the temporal dimension: accuracy decreases as the variety and volume of the full dataset is reduced.

	100%	1st Half Year	Last Half Year	Last Year
LogReg	92.70	89.86 (-2.84)	92.00 (-0.7)	92.22 (-0.48)
CNN	92.99	89.90 (-3.09)	92.19 (-0.8)	92.41 (-0.58)

level of the baseline. Complex models are more likely to overfit. For the CNN, these effects are thus stronger since less data contains less variety.

3.6.2 Leveraging Big Data through sensor proximity

Thus far the network-wide prediction was modelled naively: one predictor per sensor was trained using data only from its own history. Traffic on a particular road is influenced by traffic in its proximity, hence the predictors should model this accordingly. I therefore proceed by including data from neighbouring traffic for each predictor (still one predictor per sensor), based on the Euclidean distance between sensors, computed from geographical coordinates. This does not correspond to the actual city block distance it takes to navigate between sensors / roads. Some sensors have multiple coordinates, in which case the location is approximated to the average – a potential source of error.

The data from $k \in \{3, 5, 10\}$ neighbouring sensors is added by simply concatenating it to the input data, resulting in a training vector of length $|x_s| = w \times k$ for each predictor. This is not necessarily the most efficient or best method of performing feature selection or simultaneous multivariate prediction. A better means of determining the correlations between traffic at each sensor is to perform spatial partitioning [5] and leverage the volume traffic data itself, instead of using map coordinates. However, this results in a fixed graph representation, while correlations between roads are likely to change throughout the day (e.g. peaks in Figure 3.4). This has been observed in [84], where univariate and multivariate methods are compared, also based on map coordinates. They note that the parameters for the multivariate modeling of traffic flow are not stationary. The same observation is made in a study [32] on traffic spatio-temporal correlations where the authors conclude that the autocorrelation struc-

ture changes both spatially and temporally, according to the traffic peaks, thus a non-stationary approach is preferable.

The results are shown in Figure 3.7, where the main observation is that adding data from proximity results in better predictions. The accuracy increases to 93.24% for the CNN, in the case where the window size ($w = 10$) and data from three neighbouring sensors is used ($k = 3$), the highest accuracy recorded thus far. This result is also better than the case where $w = 20$ and $k = 0$ (for the CNN the accuracy was 93.13%, see Figure 3.5).

This implies that proximity data and feature selection have more impact than simply increasing the window size, although both are beneficial. The pattern in Figure 3.7 is clear. As more time-points and more data from neighbouring sensors are added, the performance increases. However, the result obtained using CNNs with a $w > 5$ and $k > 5$, is lower although very close to the best result obtained ($w = 10$ and $k = 3$). The effects of the curse of dimensionality thus become more evident as the length of the feature vector is increased to more than 30 time points. For logistic regression the performance still increases, although still less accurate than the CNN (a matter of regularization).

3.6.3 Performance as a function of location

The resulting test accuracies from the best logistic regression results are clustered (over all sensors) and the corresponding colour coded prototypes are overlaid on the map in Figure 3.8. From the distribution of accuracy it can be observed that prediction is more challenging in the extremities and the CBD. The geometry of road segments does not appear to have an impact on accuracy. Furthermore, there seems to be no connection between the direction of traffic (outbound vs. inbound) and accuracy, since the clusters are distributed evenly. There are 150 sensors (7.2%) where the accuracy is one standard deviation below the mean (taken over all sensors). Out of these, I examined the volume data for one sensor and observed that after the first three years the traffic volume is doubled. Thus, almost all traffic after the 3rd year is labelled as high volume. Permanent changes to the traffic rules can have a drastic impact on the distribution of the volume data for one sensor. Therefore, a threshold over the entire

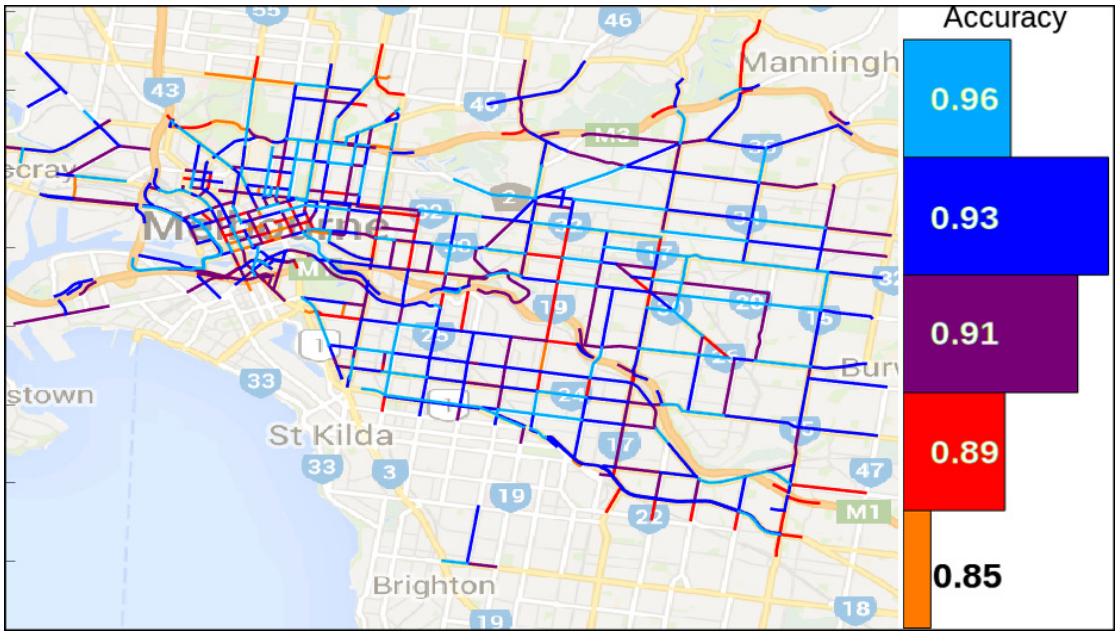


Figure 3.8: Accuracy distribution over the traffic network (logistic regression). The histogram on the right shows the relative size of each accuracy cluster.

time series, introduces additional error.

3.7 Discussion and Conclusion

In this chapter, I have explored a novel big traffic data set for the application of traffic forecasting. The classification experiments show that increasing temporal context is beneficial, provided an appropriate representation and that including neighbouring data further improves performance.

This is also consistent with the work in [102, 135] where however, the volume and complexity of the dataset is lower. Additionally, it was shown that using outdated or smaller volumes of data causes the prediction performance to drop, suggesting that the volume and variance of data is critical for Big Data applications. An additional increase in accuracy can also be obtained if the missing data is augmented with the average contextual traffic trend. The accuracy can be further increased if the goal is to forecast Mondays to Fridays only, or other subsets of data.

For useful real-time predictions, the threshold that separates high and low traffic should be adapted dynamically according to the query point (sensor) and the time of the day. Setting an appropriate threshold can be considered a separate problem in itself. The thresholding procedure introduces more problems than it solves.

Chapter 4

Topology-regularized universal vector autoregression for traffic forecasting in large urban areas

This chapter is based on the following peer-reviewed publication:

F. Schimbinschi, L. Moreira-Matias, V.X. Nguyen, J. Bailey, "*Topology-regularized universal vector autoregression for traffic forecasting in large urban areas*" in **Expert Systems with Applications** (ESWA) vol. 82, pp. 301-316, Pergamon, 2017.

Autonomous vehicles are soon to become ubiquitous in large urban areas, encompassing cities, suburbs and vast highway networks. In turn, this will bring new challenges to the existing traffic management expert systems. Concurrently, urban development is causing growth, thus changing the network structures. As such, a new generation of adaptive algorithms are needed, ones that learn in real-time, capture the multivariate nonlinear spatio-temporal dependencies and are easily adaptable to new data (e.g. weather or crowdsourced data) and changes in network structure, without having to retrain and/or redeploy the entire system.

I propose learning Topology-Regularized Universal Vector Autoregression (TRU-VAR) and exemplify deployment with causal CNNs, benchmarked against state-of-the-art autoregressive models. The multi-task learning framework pro-

duces reliable forecasts in large urban areas and is best described as scalable, versatile and accurate. Introducing constraints via a topology-designed adjacency matrix (TDAM), simultaneously reduces computational complexity while improving accuracy by capturing the non-linear spatio-temporal dependencies between timeseries. The strength of the method also resides in its redundancy through modularity and adaptability via the TDAM, which can be altered even while the system is deployed to production. The large-scale network-wide empirical evaluations on two qualitatively and quantitatively different datasets show that the method scales well and can be trained efficiently with low generalization error.

I also provide a broad review of the literature and illustrate the complex dependencies at intersections and discuss the issues of data broadcasted by road network sensors. The lowest prediction error was observed for the proposed multi-task method, TRU-VAR, which outperforms ARIMA in all cases and the equivalent single-task predictors in almost all cases for both datasets. I conclude that forecasting accuracy is heavily influenced by the TDAM, which should be tailored specifically for each dataset and network type. Further improvements are possible based on including additional data in the model, such as readings from different road metrics, weather data, calendar events, etc.

4.1 Introduction

Expert systems are at the forefront of intelligent computing and 'soft Artificial Intelligence (soft AI)'. Typically, they are seamlessly integrated in complete business solutions, making them part of the core value. In the current work I propose a system for large-area traffic forecasting, in the context of the challenges imposed by rapidly growing urban mobility networks, which is outlined in the following paragraphs. The proposed solution relies on the formulation of a powerful inference system which is combined with expert domain knowledge of the network topology, and that can be seamlessly integrated with a control schema.

Fully autonomous traffic implies an *omniscient* AI which is comprised of two expert systems, since it has to be able to both perceive and efficiently control

traffic in real time. This implies the observation of both the network state and the entities on the network. Therefore, sensing (perception) can be done via (i) passive sensors (e.g. induction loops, traffic cameras, radar) or (ii) mobile ones (e.g. Global Positioning Systems (GPS), Bluetooth, Radio Frequency Identification (RFID)). While the crowdsourced data from moving sensors (ii) can provide high-granularity data to fill accurate Origin-Destination (O-D) matrices, their penetration rate is still scarce to scale up [128].

Forecasting traffic is a function of control as well, since changing traffic rules or providing route recommendations can have an impact on the network load. However, there are factors that are not a function of control, such as human error or extreme weather conditions, which are the actual unforeseen causes of congestion. Therefore, during the transition to fully autonomous traffic control, there will be an even greater need for accurate predictions. There are also many possible intelligent applications such as a personalized copilots making real time route suggestions based on users preferences and traffic conditions, economical parking metering, agile car pooling services, all of these paving the way towards fully autonomous self driving cars. Not surprisingly, the work in simulation by Au et al. [14] has shown that semi-autonomous intersection management can greatly decrease traffic delay in mixed traffic conditions (no autonomy, regular or adaptive cruise control, or full autonomy). This is possible by linking cars in a semi-autonomous way, thus solving the congestion 'wave' problem, if most of the vehicles are semi-autonomous.

Traffic prediction will therefore become paramount as urban population is growing and autonomous vehicles will become ubiquitous for both personal and public transport as well as for industrial automation. Currently, one may argue that automatic traffic might be a self-defeating process. A common scenario might be in the case when the recommendations from a prediction expert system are identical for all users in the network. In this case, new congestions can and will be created (most vehicles take the same route), which in turn invalidate the forecasts. This is evidently caused by *poor control policies* or a lack of adequate infrastructure. Fortunately, simple solutions for both of these issues exist. The reader can refer to the following two references for each potential issue. Çolak et al. [37] formulate the control problem as a collective travel time

savings optimization problem, under a centralized routing scheme. Different quantified levels of social good (vs. greedy individual) are tweaked in order to achieve significant collective benefits. A simple (but more socially challenging) way to overcome the infrastructure problem is recommendations for car pooling as suggested by Guidotti et al. [62].

Concerning the traffic prediction literature, most research effort is focused on motorways and freeways [95, 160, 2, 74, 11, 108, 114, 185, 204, 189, 158], while other methods are only evaluated on certain weekdays and / or at particular times of the day [160, 190]. These methods usually deploy univariate statistical models that do not take into consideration all the properties that can lead to satisfactory generalization accuracy in the context of growth and automation in urban areas, namely: 1) real-time (online) learning; 2) model nonlinearity in the spatio-temporal domain; 3) low computation complexity and scalability to large networks; 4) contextual spatio-temporal multivariable regression via topological constraints; 5) versatility towards a broad set of infrastructure types (urban, suburban, freeways); 6) adaptation to changes in network structure, without full-network redeployment; 7) redundancy and customization for each series and adjacency matrix; 8) encoding time or using multi-metric data.

In the current work I address these issues and propose a multivariate traffic forecasting method that can capture spatio-temporal correlations, is redundant (fault tolerant) through modularity, adaptable (trivial to redeploy) to changing topologies of the network via its modular topology-designed adjacency matrix (TDAM). The method can be efficiently deployed over large networks of broad road type variety with low prediction error and therefore generalizes well across scopes and applications. Figure 4.12 shows that the method can predict within reasonable accuracy even up to two hours in the future – the error increases linearly and the increase rate depends on the function approximator, the TDAM and the quality of the data. I provide a comparison with state of the art methods in Table 4.1 according to properties that are essential to the next generation of intelligent expert systems for traffic forecasting:

Table 4.1: Comparison of TRU-VAR properties with state of the art traffic forecasting methods. Properties that couldn't be clearly defined as either present or absent were marked with '∼'.

Property	STARIMA[1]	DeepNN[2]	VSSVR[3]	STRE[4]	GBLS[5]	TRU-VAR[6]
1) Online learning	∼	✓	∼	✗	✗	✓
2) Nonlinear representation	✗	✓	✓	✓	✗	✓
3) Low complexity	✗	✗	✗	✓	∼	✓
4) Topological constraints	✗	✗	∼	✓	✓	✓
4) Non-static spatio-temporal	✗	✓	∼	✗	✓	✓
5) Infrastructure versatility	✗	∼	✗	∼	✓	✓
6) Easy to (re)deploy	✗	✗	✗	✗	✗	✓
7) Customizable design matrix	✗	✗	∼	∼	∼	✓
7) Distinct model per series	✗	✗	✓	✗	✗	✓
7) Transferable cross-series	✗	✗	∼	✓	∼	✓
8) Adaptable to multi-metrics	✗	✓	✓	∼	∼	✓

The contributions are as follows: (i) I propose learning Topology-Regularized Universal Vector Autoregression (TRU-VAR), a novel method that can absorb spatio-temporal dependences between multiple sensor stations; (ii) The extension of TRU-VAR to nonlinear universal function approximators over the existing state of the art machine learning algorithms, resulting in an exhaustive comparison; (iii) Evaluations performed on two large scale real world datasets, one of which is novel; (iv) Comprehensive coverage of the literature, and an exploratory analysis considering data quality, preprocessing and possible heuristics for choosing the topology-designed adjacency matrix (TDAM).

In conclusion: TRU-VAR shows promising results, scales well and is easily deployable with new sensor installations; careful choice of the adjacency matrix is necessary according to the type of dataset used; high resolution data (temporal as well as spatial) is essential; missing data should be marked in order to

[1] Kamarianakis and Prastacos [83]
[2] Lv et al. [115]
[3] Xu et al. [192]
[4] Wu et al. [190]
[5] Salamanis et al. [144]
[6] Nonlinearity dependent on the function approximator. Careful design of the topological adjacency matrix is essential and requires domain knowledge in order to define the appropriate heuristics.

distinguish it from real congestion events; given that the methods show quite different results on the two datasets I argue that a public set of large-scale benchmark datasets should be made available for testing the prediction performance of novel methods.

4.2 Related work

Traffic forecasting methodologies can be challenging to characterize and compare due to the lack of a common set of benchmarks. Despite the numerous methods that have been developed, there is yet none that is modular, design-flexible and adaptable to growing networks and changing scopes. The scope (e.g. freeway, arterial or city) and application can differ across methods. Therefore, it is not trivial to assess the overall performance of different approaches when the datasets and metrics differ. Often, subsets of the network are used for evaluating performance as opposed to the general case of network-wide prediction, which includes highways as well as suburban and urban regions. Furthermore, off-peak times and weekends are also sometimes excluded. For critical reviews of the literature the reader can follow up on these references: [132, 181, 173, 179, 154, 153].

Traffic metric types for sensor loops and floating car data: When it comes to metrics, speed, density and flow can be used as target prediction metrics. Flow (or volume) is the number of vehicles passing though a sensor per time unit (usually aggregated in 1, 5 or 15 minute intervals). Density is the number of vehicles per kilometre. It was shown [36] that multi-metric predictors can result in lower prediction error. That is, variety of input data metrics is beneficial. As to the metric being predicted, some authors argue that flow is more important due to its stability [105] while others [49] have found that traffic conditions are best described using flow and density as opposed to speed, as output metric. Nevertheless, there is a large amount of work where speed is predicted, as opposed to flow or density [144, 54, 124, 11, 85, 135, 102, 49]. This data can come from either loop sensors (two are needed) or floating car data such as that collected from mobile phones, GPS navigators, etc. For the traffic assignment problem (balancing load on the network), density is a more appropriate metric

as opposed to flow, according to the recent work in [81]. The authors make the observation that vehicles travel with a speed which is consistent with the traffic density as opposed to flow. For the current work I therefore use only *flow* data for both the independent and dependent target variables, since there were no other metrics readily available for the two datasets. I would have adopted a multi-metric approach (e.g. using speed and density data as additional input metrics) had I been able to acquire such data. However, the extension is trivial.

4.2.1 Traffic prediction methods

A comparison between parametric and non-parametric methods for single point traffic flow forecasting based on theoretical foundations [154] argues that parametric methods are more time consuming while non-parametric methods are better suited to stochastic data. An empirical study with similar objectives [86] provides a comparison of neural networks methods with statistical methods. The authors suggest a possible synergy in three areas: core model development, analysis of large data sets and causality investigation. While the focus is on short-term forecasting, it is evident that forecast accuracy degrades with a larger prediction horizons. I show that the prediction error increases linearly for the method in Figure 4.12 on page 118. The rate of increase depends on the function approximator that is being used and the design of the topology matrix. For long-term forecasting (larger prediction horizons – section 4.3, page 95), continuous state-space models such as Kalman filters [185] or recurrent neural networks (RNN) [41] have outperformed traditional 'memoryless' methods.

Parametric methods: Prediction of traffic flow using linear regression models was deployed in [110, 80, 140] while non-linear ones were applied in [73]. ARIMA [19, ch. 3-5] are parametric linear models extensively used in time series forecasting that incorporate the unobserved (hidden) variables via the MA (moving average) component. Seasonal ARIMA (SARIMA) models can be used where seasonal effects are suspected or when the availability of data is a constraint, according to the work in [99]. ARIMAX use additional exogenous data. In [187] data from upstream traffic sensors was used for predicting traffic using ARIMAX models. The results outperformed the simpler ARIMA models

at the cost doubling the computational complexity and decreased robustness to missing data. Traffic state estimation with Kalman filters was evaluated on freeways [191, 185] in combination with other methods such as discrete wavelet transforms in order to compensate for noisy data.

Non-parametric methods: One of the first applications of the K Nearest Neighbours (KNN) algorithm for short-term traffic forecasting was in [36]. KNNs have also been applied to highway incident detection [114]. The latter research made use of historical accident data and sensor loop data, representing conditions between normal and hazardous traffic. Hybrid multi-metric k-nearest neighbour regression (HMMKNN) was proposed for multi-source data fusion in [74] using upstream and downstream links.

A Support Vector Regression (SVR) model was applied to travel time prediction [189] on highways. It was compared only with a naive predictor. Online SVRs with a Gaussian kernel were deployed for continuous traffic flow prediction and learning in [198]. The method outperformed a simple neural network with one hidden layer. However, it is important to note that the neural network was trained on historical averages, which is not equivalent to training on online streaming data. Under non-recurring atypical traffic flow conditions, online SVRs have been shown to outperform other methods [24]. SVRs with Radial Basis Function (RBF) kernels showed a marginal improvement over Feed Forward Neural Networks (FFNN) and exponential smoothing [11]. The predictions were done independently for each link, thus spatial information was not leveraged. The authors clustered links according to the prediction error using K-means and Self-Organizing Maps (SOM).

Neural networks have been extensively evaluated for short-term real-time traffic prediction [35, 18, 49, 41, 102, 135, 54]. A comparison of neural networks and ARIMA in an urban setting found only a slight difference in their performance [35]. Feed forward neural networks were also used to predict flow, occupancy and speed [49]. While prediction of flow and occupancy was satisfactory, prediction of speed showed less prospects. In [135] a neural network was used for simultaneous forecasting at multiple points along a commuter's route. Multiple FFNNs were deployed (one per station) in [102] with input data from the same day, the previous week and data from neighbouring links. The weekday

was also added as an input in the form of a binary vector. This provided better spatial and temporal context to the network, thus reducing forecasting error. Time information in the form of time of day and day of week was used as additional information for traffic prediction using FFNNs in [26] also resulting in improved performance. Recurrent neural networks (RNN) demonstrated better forecasting performance [41] at larger prediction horizons compared to FFNNs, mostly due to their ability to model the unobserved variables in a continuous state space. The performance was superior when the data was aggregated into 5 minute bins (90-94%) versus 10 minutes (84%) and 15 minutes (80%). Bayesian networks were also deployed for traffic flow prediction [23]. No empirical comparison with other methods was provided.

It is important to consider that as the number of parameters increase with road network complexity and size, the parsimony of non-parametric methods becomes more evident [47].

Hybrid Methods: Existing short-term traffic forecasting systems were reviewed under a Probabilistic Graphical Model (PGM) framework in [108]. The authors also propose coupling SARIMA with either SVRs (best under congestion) and Kalman filters (best overall). This work assumes statistical independence of links. Hybrid ARIMA and FFNNs [200] were also applied to univariate time series forecasting. The residuals of a suboptimal ARIMA model were used as training data for a FFNN. No evaluations on traffic flow data was provided. A hybrid SARIMA and cell transmission model for multivariate traffic prediction was evaluated in [164]. Comparisons with other univariate or multivariate models were not provided. Generalized Autoregressive Conditional Heteroskedasticity (GARCH) and ARIMA were combined in [29]. The hybrid model did not show any advantages over the standard ARIMA, although the authors argued that the method captured the traffic characteristics more comprehensively.

An online adaptive Kalman filter was combined with a FFNN via a fuzzy rule based system (FRBS) [159] where the FRBS parameters were optimized using Meta heuristics. The combined forecasts were better than the separate models. Functional nonparametric regression (FNR) was coupled with functional data analysis (FDA) in [160] for long term traffic forecasting on one day

and one week ahead horizons. Traffic state vectors were selected based on lag autocorrelation and used as predictor data for various types of kernels. The distance function for the kernels was computed using functional principal component analysis. The method outperformed SARIMA, FFNNs and SVRs on the selected benchmark, based on a subset of weekdays (Monday, Wednesday, Friday and Saturday) for a single expressway section. Genetic algorithms (GA) were used in [1, 180] to optimize neural network architecture structure. In [1] it was applied to the structure of time-delayed neural network while in [180] the GAs were used to optimize the number of units in the hidden layer. The optimised version reached the same performance as a predefined one with less neurons.

4.2.2 Topology and spatio-temporal correlations

There are various methods that model the spatial domain as opposed to solely the temporal one. The approaches can be characterised based on the number of timeseries used as inputs and outputs for a prediction model. Regarding the inputs, the models can be either single-series (univariate) or multi-series (multivariable). In the latter case additional contemporary data can be used and the selection can be based on either topology (if known) or learned. Further categorization can be defined based on the importance of each relevant road – static or dynamic (since the dependency structure changes – see Figure 4.10). According to the outputs, the algorithms can be single task, in which case one predictor is learned for each station and multi-task in which case parameters are coupled and predictions are made simultaneously at all stations.

Single task & multi-series: Most common among the multi-series methods is to take into consideration the upstream highway links. Kalman filters using additional data from upstream sensors have been found to be superior to simple univariate ARIMA models [158]. Data from only five sequential locations along a major 3 lane per direction arterial on the periphery was used (distance between sensors varied between 250 and 2500 meters). The authors also conclude that short-term traffic flow prediction at urban arterials is more challenging than on freeways. A similar conclusion is drawn in [180].

Spatio-temporal HMM were used for estimating travel cost, also considering spatio-temporal correlations when predicting traffic [193]. Markov random fields (MRF) were deployed to capture dependencies between adjacent sensor loops via a heat map of the spatio-temporal domain [95]. Focus was on the dependencies between the query location and the first and second upstream link connections on freeways with degree higher than one. Such upstream bifurcations were referred to as 'cones'. Data between the 'cones' and the query was not considered. The authors quantized data into twelve levels. The weights for the spatial parameters were re-estimated monthly. In more complex networks such as urban areas, these weights can change during the course of one day. Similarly, dependencies on the cliques are estimated in [2] using either multiple linear regression and SVRs, the latter resulting in better accuracy. No comparisons were made with other methods. Multivariate Adaptive Regression Splines (MARS) were used for selecting the relevant links' data and SVRs as the predictor component in [192]. The method was evaluated on a sub-area and compared to AR, MARS, SVRs, SARIMA and ST-BMARS (spatio-temporal Bayesian MARS) showing promising results. An interesting approach was the work in [124] where the core idea was to minimize execution time by transferring learned representations (on a subset of links) to the overall network. A low dimensional network representation was learned via matrix decomposition and this subset of links was used to extrapolate predictions over the entire network. The computations were sped up 10 times at the expense of increasing error by 3%.

Spatio-temporal random effects (STRE) was proposed in [190]. The algorithm reduced computational complexity over spatio-temporal Kalman filter (STKF). Data around a query point was selected and weighted using a fixed precomputed set of rules, depending on the relative position (e.g. upstream, downstream, perpendicular, opposite lane, etc). Training was done using 5 minute resolution data from Tuesdays, Wednesdays and Thursdays. Only data from 6a.m. to 9p.m. (peak hour) was considered. Predictions were made on a group of query points from two separate areas (mall area and non mall area). The authors hypothesized that the mall area could have had more chaotic traffic patterns. STRE had lower error relative to ARIMA, STARIMA and FFNNs

except for one case, the westbound non-mall area. It is important to note that the FFNNs were trained in univariate, one network per station mode. Furthermore, it is also not clear whether the prediction results were for out of sample data. Similar work making use of sensor location data, spatio-temporal random fields (STRF) followed by Gaussian process regressors were proposed in [107] for route planning.

FFNNs and Radial Basis Function Networks (RBFNs) were combined into the Bayesian Combined Neural Network (BCNN) [204] model for traffic flow prediction on a 15 minute resolution highway dataset. Data from the immediate and two links upstream sensors as well as downstream sensors was used as input to both networks. The predictions were combined linearly and weighted according to the spread of the error in previous time steps on all relevant links. The combined model was better than the individual predictors, however the RMSE was not reported nor any comparisons with other baseline methods given. Bayesian networks with SARIMA as an a priori estimator (BN-SARIMA) and FFNNs as well as Nonlinear AutoRegressive neural network with eXogenous inputs (NARX) were compared in [54] for floating car data. The learning architectures used data from the output and conditioning links and predictions were single task. The results were marginally different for a subsection of reliable data. For network-wide forecasting on both 5 and 15 minute intervals, NARX and FFNNs were better than BN-SARIMA.

Multi-task & multi-series: Spatio-Temporal ARIMA (STARIMA) [83] were perhaps the first successful traffic forecasting models that focused on the spatio-temporal correlation structure. However, the spatial correlations were fixed, depending solely on the distances between links. I empirically show in Figure 4.10 (Page 113) that the spatial correlations can change during the course of a day. STARIMA compensate for non-stationarity by differencing using the previous day values which can bias the estimated autoregression parameters (traffic behaviour can be considerably different e.g. Monday vs. Sunday). A study on autocorrelation on spatio-temporal data [32] concludes that ARIMA based models assume a globally stationary spatio-temporal autocorrelation structure and thus are insufficient at capturing the changing importance between prediction tasks. The work in [121] addresses this problem using Dynamic Turn Ratio

Prediction (DTRP) to update the normally static matrix containing the structural information of the road network. Under the hypothesis that the physical distance between road sections (tasks) does not accurately describe the importance of each task, a VARMA [113] model is refined by Min et al. [120] to include dependency among observations from neighbouring locations by means of several spatial correlation matrices (as many as the number of lags). In one matrix, only related tasks are non-zero. The authors do not explicitly define task relatedness, however most likely all upstream connections are selected. Further sparsity is introduced by removing upstream links, under the hypothesis that such links are unlikely to influence traffic at the query task, given the average travel speed for that location and time of day. There could be one such matrix for peak times and one for off-peak times, depending on the design choice. Another ARIMA inspired algorithm [85] makes use of a parametric, space-time autoregressive threshold algorithm for forecasting velocity. The equations are independent and incorporate the MA (moving average) and a neighbourhood component that adds information from sensors in close proximity, based on the Least Absolute Shrinkage and Selection Operator (LASSO) [169].

Building up on the previously mentioned work and avoiding the potentially problematic thresholds used to to identify the discrete set of regimes, a Graph Based Lag STARIMA (GBLS) [144] model was trained using speed data from GPS sensors, with the goal of travel time prediction. Unlike the previous work, the graph structure was initially refined using a breadth-first search based on degree (number of hops). For the selected connections, spatial weights were computed and used in the STARIMA model. The weight matrix for each lag was fixed and contained the inverse of the lag-sums Pearson correlations between relevant roads. Finally, the model took into consideration the current speed on the road, the previous two speed values and the average speed of the top 10 ranked relevant roads. The introduced sparsity reduced computational complexity. However, from the current experiments it can be observed that the correlations are typically nonlinear, which implies that this measure is not appropriate for ranking. The data used was only for the same day of the week for both training and testing for the two datasets. GBLS was compared with univariate KNNs, Random Forests (RF), SVRs and Compressed SVRs. No

comparison with FFNNs were made. The behaviour of the proposed algorithm was very different over the two datasets, however the proposed method had lower prediction error then the benchmarks. It is also not clear whether the parameters were coupled or not. One could argue that this class of methods do not follow the law of parsimony (Occam's razor) – there are too many design choices, assumptions and parameters.

Recently FFNNs with many hidden layers (deep learning) have been applied to network wide traffic prediction on highway data [115]. While neural network based models were able to learn the nonlinear spatio-temporal correlations, this type of approach – as well as the similar linear STARIMA class of models [85, 120, 121, 83] – does not explicitly leverage the topological structure. Very recently, there have been introduced a few new, advanced RNNs which consider the topology of a road network to predict traffic, where the topology is integrated within the RNNs [106, 77, 76]. The latter are very related to the TRU-VAR framework since causal CNNs can be replaced with RNNs, effectively arriving at very similar results.

Hence it is likely that prediction error can be further reduced by leveraging this information explicitly. Conversely to the aforementioned work, in Topological Vector Autoregression (TRU-VAR), the relative importance of the related timeseries is adjusted automatically, also accounting for the contextual time.

In summary, the following observations can help improve traffic prediction performance: (**1**) nonlinearity is important for explaining traffic behaviour; (**2**) leveraging the topological structure could result in lower errors; (**3**) the system should be flexible to changes in the adjacency matrix design; (**4**) static spatio-temporal models are not appropriate for complex urban roads; (**5**) simplicity (Occam's razor) is to be preferred in general [47]; (**6**) multi-metric data can increase accuracy, speed data is to be avoided as a target metric [36]; (**7**) for certain models, explicitly encoding time can decrease error; (**8**) crowdsourced data from vehicles can help reduce sensor noise and provide redundancy to missing data; (**9**) continuous state-space models are more adequate at capturing complex patterns and therefore making predictions due to the highly dynamic nature of driver behaviour.

Considering all of the above, in the next section I introduce the theoretical

motivations for TRU-VAR. I start with the extension of VAR to topological constraints, continue the generalization with examples of function approximators according to state of the art prediction models and conclude with the optimization process.

4.3 Topological vector autoregression

Traffic prediction in large urban areas can be formulated as a multivariate timeseries forecasting problem. Vector AutoRegression (VAR) is a natural choice for such problems. Since in this particular case the precise location of each sensor station is also provided, the topological structure can also be leveraged and used as prior information in order to constrain the number of parameters that are to be estimated. In effect this reduces the computational complexity and improves accuracy over simple univariate forecasting methods, while learning in real time the spatio-temporal correlations between the contemporary timeseries.

From a high level perspective the idea is simple: I assume that for a particular prediction station (timeseries) it is beneficial to use data from stations in close proximity (contemporary), however I exclude stations that are distant. This induces sparsity since the data from all other sensor stations does not have to be used, while capturing the spatio-temporal dynamics. I provide empirical arguments in an exploratory analysis of what the distance heuristics can be defined as in section 4.4 where I also show that the spatio-temporal correlations change throughout the day.

Given a multivariate timeseries dataset $\tilde{X} \in \mathbb{R}^{T \times S}$ with T non i.i.d. observations over S series, I denote $\boldsymbol{x}^s_{\Delta t} = \begin{bmatrix} 1 & x^s_t & x^s_{t-1} & \dots & x^s_{t-\Delta} \end{bmatrix}^\top$ as the predictor vector for sensor s consisting of the past observations of length Δ. The corresponding response variable $y^s_{t+h} = x^s_{t+h}$ is the value to be predicted h time steps ahead. Prediction can then be modelled as follows:

$$y^s_{t+h} = f(\boldsymbol{x}^s_{\Delta t}, \boldsymbol{\theta}^s) + \epsilon^s_{t+h} \quad \text{where } \epsilon \in \mathcal{N}(0, \sigma) \tag{4.1}$$

The prediction horizon h indicates how far in the future predictions are

made. For short-term forecasting this is typically the immediate time step (e.g. $h = 1$) in the future. The first element of the input vectors is set to 1 to indicate the intercept constant. In the time series forecasting literature Δ is more commonly known as the *lag* or the number of previous observations from the past that are used to make predictions. For neural networks this is sometimes referred to as the receptive field. To satisfy the assumption that $\mu_\epsilon = 0$ and normally distributed, I later discuss differencing. If f is the dot product, then Equation 4.1 describes an autoregressive model (AR(Δ)) of order Δ. AR models operate under the assumption that the errors are not autocorrelated. However, the errors are still unbiased even if they are autocorrelated. ARIMA models also incorporate a Moving average (MA). These are identical with the difference that predictions are made based on the past prediction errors $\boldsymbol{e}^s_{\Delta t}$ where $e^s = \hat{y}^s - y^s$. It is possible to write any stationary AR(Δ) model as an MA(∞) model.

4.3.1 Topology Regularized Universal Vector Autoregression

In multivariable regression models, additional data can be used for the predictor data such as contemporary data and / or an encoding of the time of day $\boldsymbol{z}_{\Delta t}$. Any other source of relevant data, such as weather data, can also be used as input.

Vector AutoRegression (VAR) models [64, ch. 11] are a generalization of univariate autoregressive models for forecasting multiple contemporary time-series. I refer the reader to [13] for a discussion on VARs in contrast to VARMAs. VARs are not a closed class when the data is aggregated over time like VARMAs are. However, VARs have been found to be often good approximations to VARMAs, provided that the lag is sufficiently large [113]. I start with a two dimensional VAR(1) with one lag, where $\boldsymbol{\theta}$ (Equation 4.1) is the vector containing the autoregressive parameters consisting of θ^{ij}_t which weight the external spatio-temporal autoregressive variables, specifically the influence of the t-th lag of series x^j on series x^i:

$$
\begin{aligned}
y^1_{t+h} &= \theta^1_0 + \theta^{11}_t x^1_t + \theta^{12}_t x^2_t + \ldots + \theta^{1S}_t x^S_t + \epsilon^1_{t+h} \\
y^2_{t+h} &= \theta^2_0 + \theta^{21}_t x^1_t + \theta^{22}_t x^2_t + \ldots + \theta^{2S}_t x^S_t + \epsilon^2_{t+h} \\
&\ldots \\
y^s_{t+h} &= \theta^s_0 + \theta^{s1}_t x^s_t + \theta^{s2}_t x^2_t + \ldots + \theta^{sS}_t x^S_t + \epsilon^s_{t+h}
\end{aligned}
$$

It becomes evident from the above equations that the required number of parameters to be estimated increases quadratically with the number of timeseries. Furthermore, if more lags are added, the computational complexity becomes $O((s \times l)^2)$. Therefore, introducing sparsity is both beneficial (Equation 4.2) and necessary since it introduces constraints on the number parameters that are to be estimated. Sparsity refers to reducing the number contemporary time series (and therefore parameters to be estimated) that are relevant for a query location. Since the topological structure of road networks is known, introducing such priors is intuitive since the relevance of contemporary timeseries does vary. I therefore introduce sparse topology regularization for vector autoregression (TRU-VAR) based on the road geometry via a topology-designed adjacency matrix $A \in \{0, 1\}$. If G is the graph describing road connections, then I denote $G^1(i)$ to be the set of all first order graph connections with node i, then:

$$
a^{ij} = \begin{cases} 1, & i = j; \\ 0, & j \notin G^1(i); \\ 1, & j \in G^1(i). \end{cases}
$$

Evidently, I could have also opted to select higher order degrees e.g. $G^2(i)$, for designing the topological adjacency matrix, however in urban settings the number of such neighbours can increase exponentially as the degree increases. However, the matrix could be heuristically adapted according to the number of first order connections, in the case of highways where the degree of a node can be low. Other means of defining A can be used such as ranking based on correlation of the timeseries. However, it can be observed in Figure 4.9 (plotted using Google Maps) that correlation does not necessarily correspond to the relevant

topology. In contrast, in this case the relevant importances are learned through fitting the model parameters, thus capturing the spatio-temporal dynamics. The sparsity terms are introduced and written in general form:

$$y^s_{t+h} = \theta^s_0 + \sum_{k=1}^{S} a^{sk}\, \theta^{sk}_t\, x^k_t + \epsilon^s_{t+h} \tag{4.2}$$

The previous equation can then be generalized for one series to multiple lags:

$$y^s_{t+h} = \theta^s_0 + \sum_{k=1}^{S} a^{sk} \sum_{\delta=0}^{\Delta} \theta^{sk}_{t-\delta}\, x^k_{t-\delta} + \epsilon^s_{t+h} \tag{4.3}$$

Finally, written in compact form and generalized for any function approximator:

$$\begin{aligned} y^s_{t+h} &= \boldsymbol{\theta}^{s\top} \boldsymbol{x}^s_{\Delta t} \odot [(\boldsymbol{a}^s)_{\times\Delta}] + \epsilon^s_{t+h} \\ y^s_{t+h} &= f\left(\boldsymbol{x}^s_{\Delta t} \odot [(\boldsymbol{a}^s)_{\times\Delta}], \boldsymbol{\theta}^s\right) + \epsilon^s_{t+h} \end{aligned} \tag{4.4}$$

For convenience, the inputs can be stacked (predictor data) into a matrix with $N = T - \Delta$ rows and $M = \Delta$ columns $X^s \in \mathbb{R}^{N\times M}$ and the outputs (response data) into the vector $\boldsymbol{y}^s \in \mathbb{R}^N$. I denote the lag constructed matrices for all series as $X = \begin{bmatrix} X^1 & X^2 & \dots & X^S \end{bmatrix}$ and $Y = \begin{bmatrix} \boldsymbol{y}^1 & \boldsymbol{y}^2 & \dots & \boldsymbol{y}^S \end{bmatrix}$. Finally, I define $A^s = \begin{bmatrix} (\boldsymbol{a}^s)_{\times\Delta} \end{bmatrix}$ and $A = \begin{bmatrix} A^1 & A^2 & \dots & A^S \end{bmatrix}$.

$$\boldsymbol{y}^s = f\left(X^s \odot A^s, \boldsymbol{\theta}^s\right) + \boldsymbol{\epsilon}^s \tag{4.5}$$

Note that from here on the Hadamard product with the sparsity inducing matrix A is omitted for brevity.

4.3.2 Structural Risk Minimization

The assumption is made that there is a joint probability distribution $P(x, y)$ over X^s and $\boldsymbol{y}^s$ which allows us to model uncertainty in predictions. The risk cannot be computed in general since the distribution $P(x, y)$ is unknown. The empiri-

cal risk is an approximation computed by averaging the loss function L on the training set. In Structural Risk Minimization (SRM) [174] a penalty Ω is added. Then, the learning algorithm defined by SRM consists in solving an optimization problem where the goal is to find the optimal fitting parameters $\hat{\boldsymbol{\theta}}^s$ that minimize the following objective:

$$\hat{\boldsymbol{\theta}}^s = \underset{\theta^s}{\operatorname{argmin}} \Big(L\big(f(X^s, \boldsymbol{\theta}^s), \boldsymbol{y}^s\big) + \lambda^s \Omega(\boldsymbol{\theta}^s) \Big) \tag{4.6}$$

For regression, under the assumption of normally distributed errors, the mean squared error (MSE) or quadratic loss is commonly used since it is symmetric and continuous. Thus, in least squares minimizing the MSE (Equation 4.7) results in minimizing the variance of an unbiased estimator. As opposed to other applications, in the current problem setting it is desirable to put a heavier weight on larger errors since this maximizes the information gain for the learner [27]. It is important to mention that the RMSE is reported since it is linked to the loss function, however the MAPE is also reported, which is specific to traffic forecasting literature. For a comprehensive discussion on predictive accuracy measures the reader is referred to [42].

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2 = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2 \tag{4.7}$$

4.3.3 Regularized Least Squares

Since the data consists of previous observations in time it is possible that these could have a varying degree of importance to the predictions, and furthermore these could be different for each data stream. Using the quadratic error it is easy to arrive at the least squares formulation in Equation 4.8 where Ω is a penalty on the complexity of the loss function which places bounds on the vector space norm. Since the choice of the model can be arbitrary, regularization can be used to improve the generalization error of the learned model. With a lack of bounds on the real error these parameters are tuned using the surrogate error from a validation dataset or using cross-validation.

For a negative log likelihood loss function, the Maximum a Posteriori (MAP)

solution to linear regression leads to regularized solutions, where the prior distribution acts as the regularizer. Thus, a Gaussian prior on $\boldsymbol{\theta}$ regularizes the L_2 norm of $\boldsymbol{\theta}$ while a Laplace prior regularizes the L_1 norm [see 129, ch. 5-9]. ElasticNet [205] is a combination of L_1 and L_2 regularization which enforces sparsity on groups of columns such as the ones in X where a parameter α balances the two lambdas of the two norms. In effect, ElasticNet generalizes both L_1 and L_2 regularization. I experiment with both regularization methods.

$$\sum_{s=1}^{S} \min_{\theta^1,\ldots,\theta^S} \|f(X^s, \boldsymbol{\theta}) - \boldsymbol{y}\|^2 + \lambda_2 \|\boldsymbol{\theta}\|^2 + \lambda_1 \|\boldsymbol{\theta}\|_1 \tag{4.8}$$

4.3.4 The function approximator model

Having defined the optimization problem, loss function and regularization prior types for one timeseries I can now consider the function approximation model f^s, starting with the simple linear models.

Linear least squares If the combination of features is linear in the parameter space $\boldsymbol{\theta}$, then the regression model is linear. Typically the error ϵ is assumed to be normally distributed. Any non-linear transformation $\phi(\boldsymbol{x})$ of the input data X such as a polynomial combination of features thus still results in a linear model. In the current experiments a simple linear combination of features is used ($\phi(x) = x$) for linear least squares (LLS):

$$f(X, \boldsymbol{\theta}) = \sum_{m=1}^{M} \boldsymbol{\theta}_m \phi(x)_m \tag{4.9}$$

By replacing f with Equation 4.9 in Equation 4.8, and setting $\lambda_1 = 0$, the parameters $\boldsymbol{\theta}^s$ can be found analytically for ridge regression using the normal equation:

$$\hat{\boldsymbol{\theta}} = (X^\top X + \lambda_2 I)^{-1} X^\top \boldsymbol{y} \tag{4.10}$$

For ordinary least squares (OLS) where regularization is absent we can simply set $\lambda_2 = 0$. It is important to note that the solution is only well defined

if the columns of X are linearly independent e.g. X has full column rank and $(X^\top X)^{-1}$ exists. Furthermore, closed form solutions (offline / batch learning) are more accurate than online learning since if the solution exists we are guaranteed to converge to the global optimum, while for online methods there is no such guarantee. In real world scenarios however, data comes in streams and offline learning is not a practical option. Real-time learning is essential to incident prediction [127]. Then, $\boldsymbol{\theta}$ can be transferred to an online learner as an initialization where the learning can continue using an online method such as stochastic gradient descent (SGD).

Kernel least squares Kernel methods such as support vector regression (SVR) [155, 162] rely precisely on non-linear transformations of the input data, usually into higher dimensional reproducing kernel Hilbert space (RKHS), where the transformed data $\phi(\boldsymbol{x})$ allows for lower generalization error by finding a better model hypothesis. In these experiments, a linear kernel is used for the support vector regression (SVR) or kernel least squares (KLS).

$$\phi : K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle \tag{4.11}$$

Using the kernel the model can be expressed as a kernel expansion where $\alpha(n) \in \mathbb{R}$ are the expansion coefficients. This transforms the model to:

$$f(\boldsymbol{x}, K) = \sum_{n=1}^{N} \alpha(n) K(x_n, x) \tag{4.12}$$

Which in turn can be formulated as kernel ridge regression:

$$\phi(\hat{\boldsymbol{\theta}}) = \underset{\phi(\theta)}{\operatorname{argmin}} \|\boldsymbol{y} - \phi(X)\phi(\boldsymbol{\theta})\|^2 + \lambda \|\phi(\boldsymbol{\theta})\|^2 \tag{4.13}$$

In practice this implies computing the dot product of the transformed input matrix which is very memory intensive. Instead, in the current experiments L-BFGS [131] is used for ridge or SPARSA [188] for LASSO.

Nonlinear least squares Multi-Layer Perceptrons with one hidden layer are a form of non-linear regression. Such models with a finite set of hidden sigmoid

activation functions are universal function approximators [39]. The simplest form can be defined using a single hidden layer model with H units, using a sigmoid activation function $\phi(x) = 1/(1+e^{-x})$ The model is reminiscent of nested kernels:

$$f(X,\Theta) = \sum_{h=1}^{H} \boldsymbol{\theta}_h \phi\Big(\sum_{m=1}^{M} \boldsymbol{\theta}_m \phi(x)_m \Big) \tag{4.14}$$

For non-linear least squares there is no closed form solution since the derivatives are functions of both the independent variable and the parameters. Such problems can be solved using gradient descent. After specifying initial values for $\boldsymbol{\theta}$ (which can also come from an offline learner), the parameters are found iteratively through successive approximation. Multiple passes are done through the dataset. In one pass, mini-batches or single examples are shown and the parameters are adjusted slightly (according to a learning rate) in order to minimize the loss.

Causal convolutional neural networks As demonstrated in subsection 2.1.3 any autoregressive model can be considered a causal model, specifically any FFNN can be considered as a causal Convolutional Neural Network (CNN). While CNNs have many parameters, here I only consider a very simple special case where only one set of filters is learned, the stride is 1 and the filter width is equal to Δ. The parsimonious approach turns out to work well in practice in this case. To summarise, the in the current work, the terms FFNN, MLP and CNN are interchangeable and equivalent.

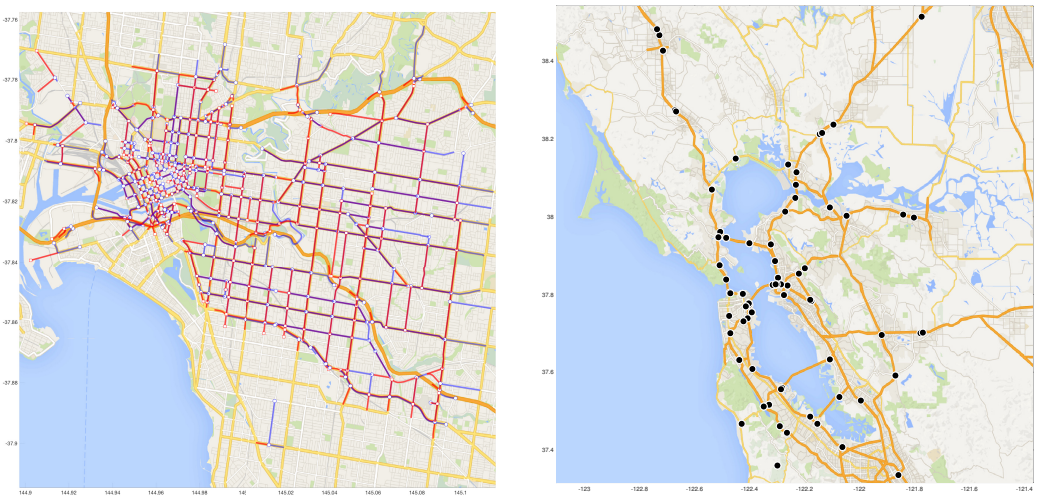
Once the TDAM is defined, TRU-VAR can therefore be generalized to any type of autoregressive (or causal) function approximator. The modularity and flexibility results in low computational complexity thus resulting in scalable fitting of nonlinear models which capture the spatio-temporal correlations. This also provides redundancy and versatility towards a broad set of road network types covering urban, suburban and freeways. Finally when the network structure changes the TDAM can be adjusted partially, which does not require redeployment of the entire prediction system.

4.4 An exploratory analysis of traffic data

In this section I discuss data. I make observations and apply the findings in the experimental section. I consider aspects such as trends, seasonality, outliers, stationarity, variable (sensor station) interdependency and data quality. I show that there are specific dependencies between sensors (Figure 4.8) which also change as a function of time (Figure 4.10).

4.4.1 Datasets

The VicRoads dataset was recorded over 6 years in the City of Melbourne, Australia and consists of volume readings from 1084 sensors covering both urban and suburban areas as well as freeways. The frequency of recordings is 15 minutes (96 readings over 24 hours). Coverage area depicted in Figure 4.1a.



(a) Melbourne roads with available traffic data are highlighted in either red or blue according to direction.

(b) PeMS station points are marked with a dot.

Figure 4.1: Schematic illustration of the sensor location for both datasets.

The California Freeway Performance Measurement System (PeMS) dataset [175] (Figure 4.1b) has been extensively used in the literature and consists of volume readings taken every 5 minutes from a network of 117 sensors over 8

months in the year 2013. As it can be seen from Figure 4.1b it consists of mostly freeway commuting and thus does not capture the complexities of inner city commuting.

I further describe the two datasets with an emphasis on VicRoads, since it is a new dataset and PeMS is well known and more studied in the literature [115, 108, and others].

4.4.2 VicRoads data quality and congestion events

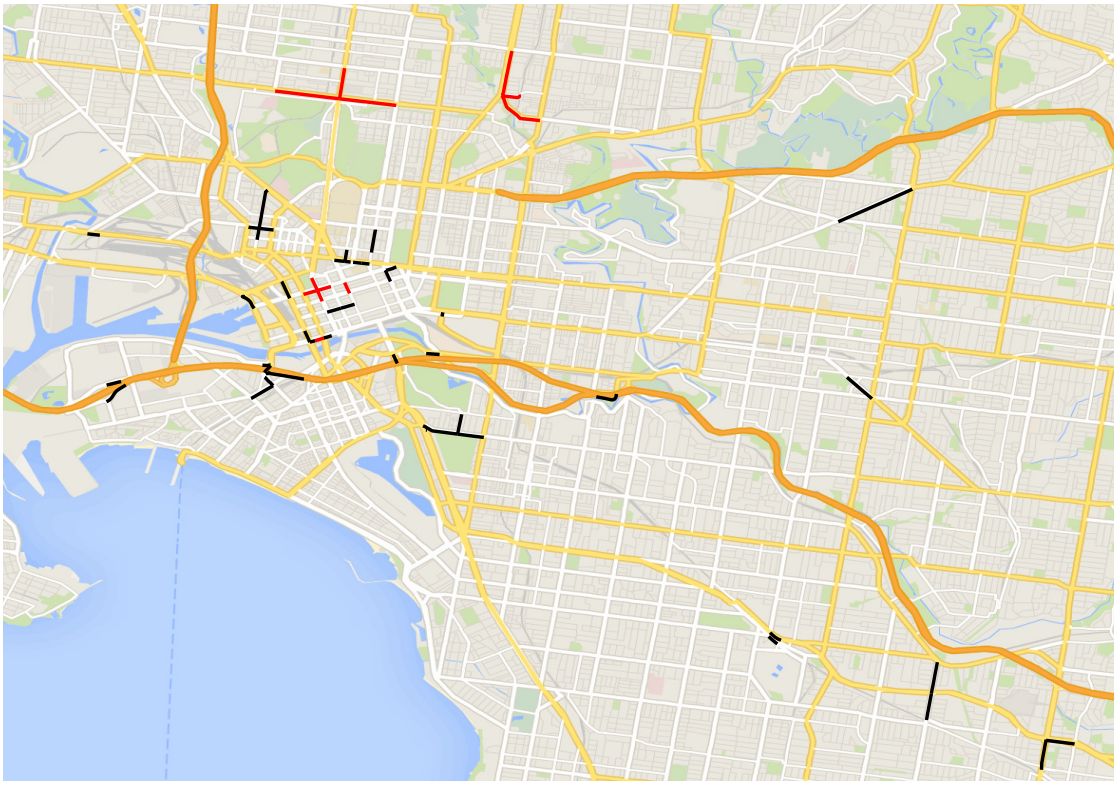


Figure 4.2: Sensors above the 95 (black) and 99 (red) quantile after discarding missing data, road sections with many congestion events.

Datasets recording traffic volume (flow) consist of positive integers in original format. There is no explicit distinction made (no labels) between congestion and missing data. Intuitively, volume is close to zero when traffic slows down and zero when it completely stops. While it is certainly possible to have zero volume in a period of 5 or 15 minutes, it is highly unlikely that this can happen

in practice and very unlikely to happen for an entire day. I proceed to mark the missing data by assuming that if the total volume for one day is zero (no cars have passed through the sensor in 24 hours) then it does not correspond to a congestion event.

I therefore use this information to discard the marked days *only from the test set*. This is important, since it allows the learner to identify congestions. Considering that in real scenarios these sensors can break down, the aim is to emphasize robustness towards such events or congestion and as such do *not* replace the missing values with averages or use any other method of inferring the missing values.

Following this operation it can be observed (Figure 4.3) that these sensors have not all been installed at the same time. Furthermore, we can also observe network-wide missing data (vertical lines). These sporadic 0 readings are not taken into consideration, since these could correspond to sudden traffic congestion, although it is still highly unlikely. However, I could have considered heuristic rules where for example, 4 consecutive readings would correspond to sensor failure (or road works) since it is very unlikely that no cars would pass within one hour through a section, even in congestion conditions. This approach was not taken.

After marking the missing data, I further compute the remaining number of 0-valued recordings for each sensor and plot in Figure 4.2 the roads above the 0.95 (black) and 0.99 (red) quantile on the map. It is quite probable that the error on these particular sensors will be higher than others, since half the data can be available for these sensor stations.

4.4.3 Seasonality, Trends, Cycles and Dependencies

It is trivial to relate to daily traffic patterns, especially peak hours. These patters are mostly stable and are a function of location and day of the week. These flows vary along the different spatio-temporal seasonalities of drivers behaviour. I adopt an initial explorative approach towards determining the statistical properties of the multivariate timeseries. I did not perform a Box-Cox transformation since upon inspection there was no evidence of changing variance. Fur-

thermore, upon visual inspection of the series it was evident that the data is non-stationary as the volume moves up and down as a function of the time of the day.

Figure 4.4 depicts the summary statistics plotted per time bin in one day, where the left figure corresponds to a typical suburban region while the right one corresponds to a highway off-ramp. The right one is typical for a road with high outbound traffic in the evening peak hours when people commute back home. Comparing these two locations allows us to get insights towards the dynamics of the different types of traffic within the network. Their location is also shown on the map in Figure 4.7.

4.4.4 Autocorrelation profiles

Figure 4.5 depicts the autocorrelation profiles of the corresponding traffic flow time series for 2 different sensors from the VicRoads dataset. Their location on the map is shown in Figure 4.7. I chose a lag corresponding to 4 days to observe

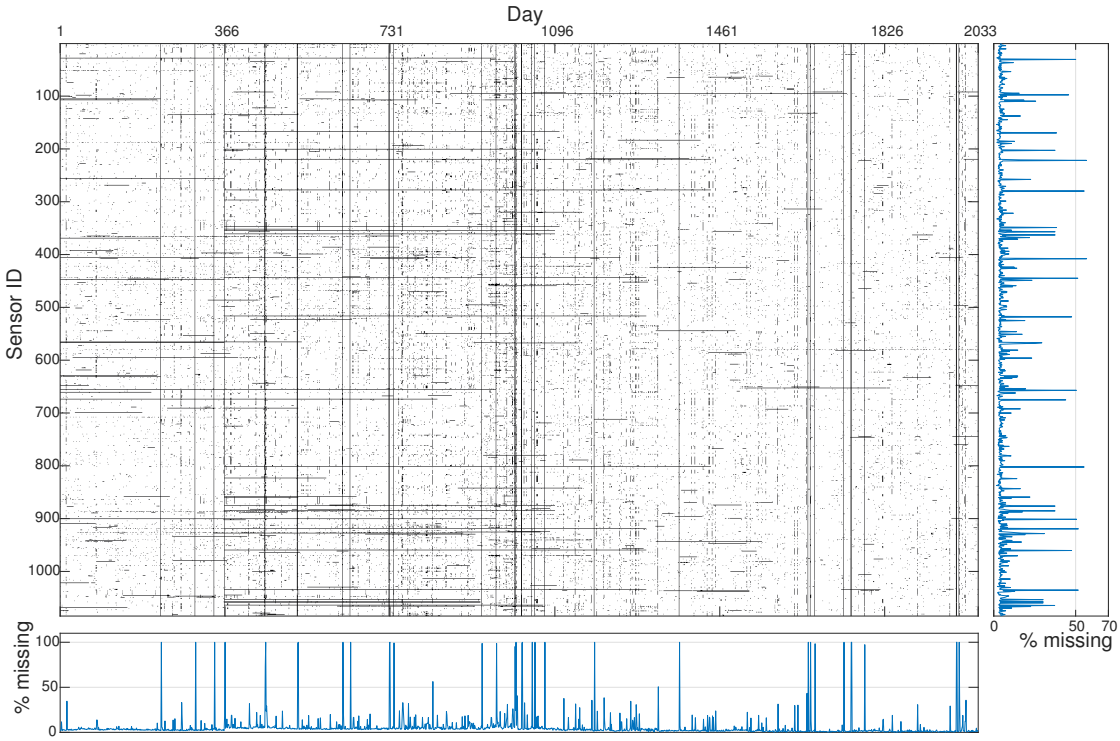
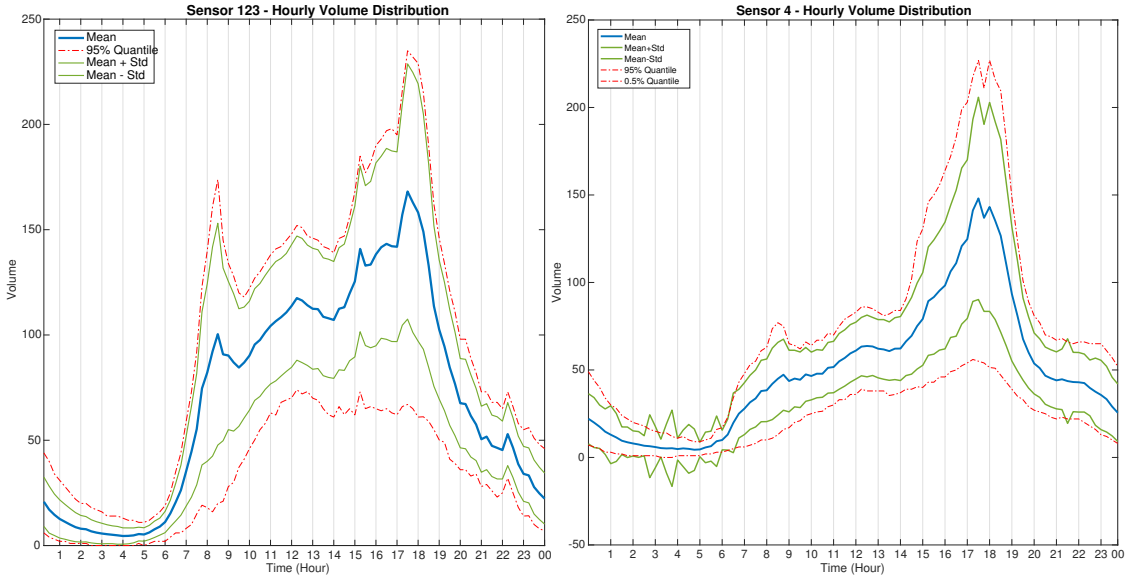


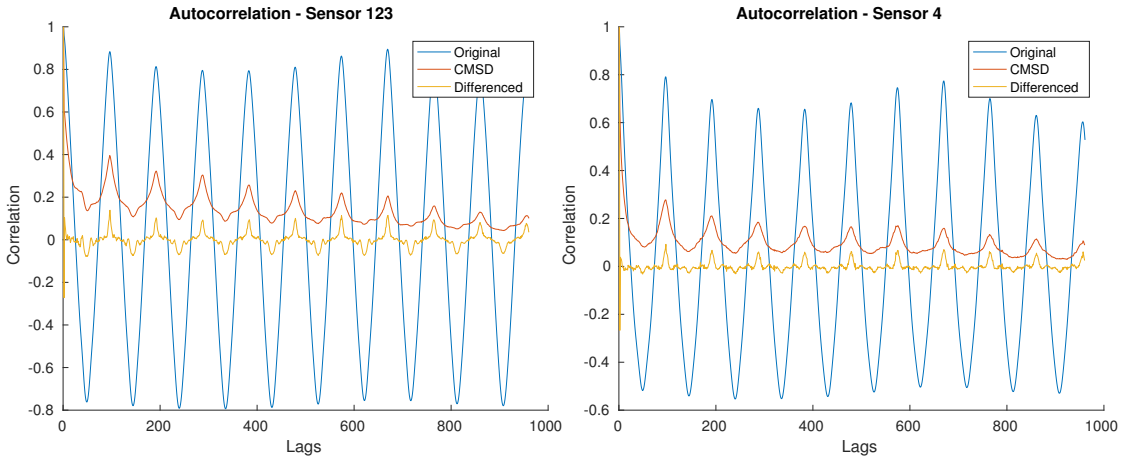
Figure 4.3: Black pixels indicate days with no readings.



(a) Sensor 123 **(b)** Sensor 4

Figure 4.4: The daily summary statistics differ for each road segment (VicRoads).

any seasonal or cyclic patterns left in the signal, after subtracting the seasonal component or differencing.



(a) Sensor 123 - Suburb **(b)** Sensor 4 - Highway off ramp

Figure 4.5: Autocorrelation plot for 400 lags. Differencing removes seasonality patterns. Daily seasonality is clearly observable.

I estimate the seasonal component for each series by computing the minute / hourly averages, separately for each day of the week, for each series. In order to

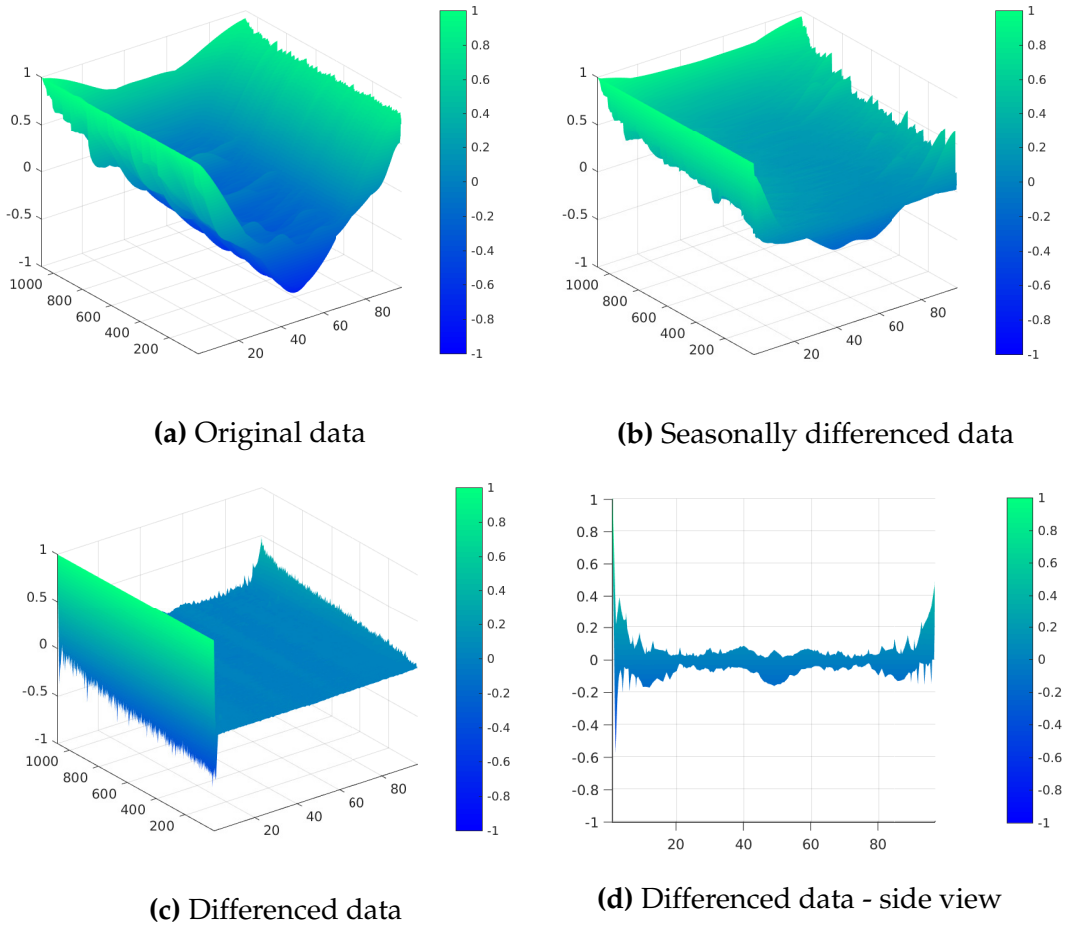
compute the averages I convert the time series to a tensor matrix. $X_\mu \in \mathbb{R}^{S \times D \times H}$ where $S = 1084$ is the number of road segments $D = 7$ is the number of days in a week and $H = 96$ is the 15 minute average number of observations made in one day. I then convert the matrix back to a timeseries and subtract this from the original time series, obtaining the contextual mean seasonally differenced (CMSD) timeseries. This operation is normally performed using a moving average. However, this way I can arrive at a more precise estimation, also as a function of the day of the week. I also differentiate each sensors' timeseries and plot the autocorrelation again for these two sensors. This method removes most of the seasonal patterns in the data.

It can be observed from Figure 4.5 that while these operations largely removes the seasonality, it is quite likely that the seasonal or trend / cyclic components could come from the (non) linear interactions with the neighbouring roads. This suggests that using proximity data is more likely to lead to increased accuracy. I further inspect the overall autocorrelation over the entire network. Hence I plot the autocorrelation for all sensors for 96 lags on the same type of data transformations. It is observable from Figure 4.6d that the seasonal components are removed for almost the entire network data.

4.4.5 Intersection correlations

While I have considered the statistical properties of each individual road section I further examine the information carried between road sections, since most roads are highly dependent on the connected or neighbouring roads. In Figure 4.7 the roads with the same direction as the query road (solid black) are marked as black and the opposite direction are marked as red dashed lines. It is important to point out that this might not be entirely accurate and depends on the convention used when marking direction (e.g. 1 is always road heading north or west).

From Figure 4.8a high correlation can be observed between the red dotted segments and the black target road (123) - implying a correlation between (apparently) opposite traffic directions, which is unlikely. For prediction, this is not crucial since all connected sections are selected. However, the major point



(a) Original data

(b) Seasonally differenced data

(c) Differenced data

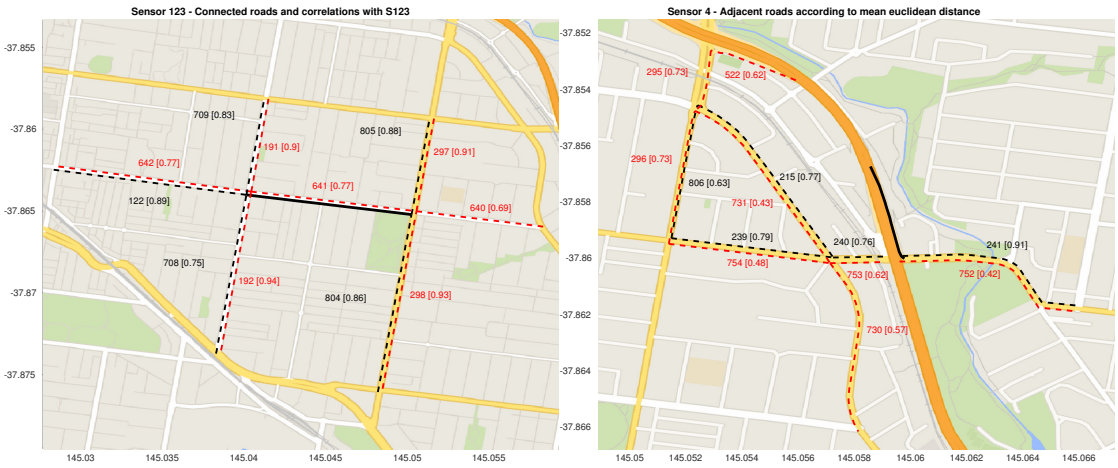
(d) Differenced data - side view

Figure 4.6: Network Wide Autocorrelation Surface.

is that there are complex dependencies at intersections. It is important to note that Pearson's correlation coefficient captures only linear correlations and there could be nonlinear correlations between two road sections.

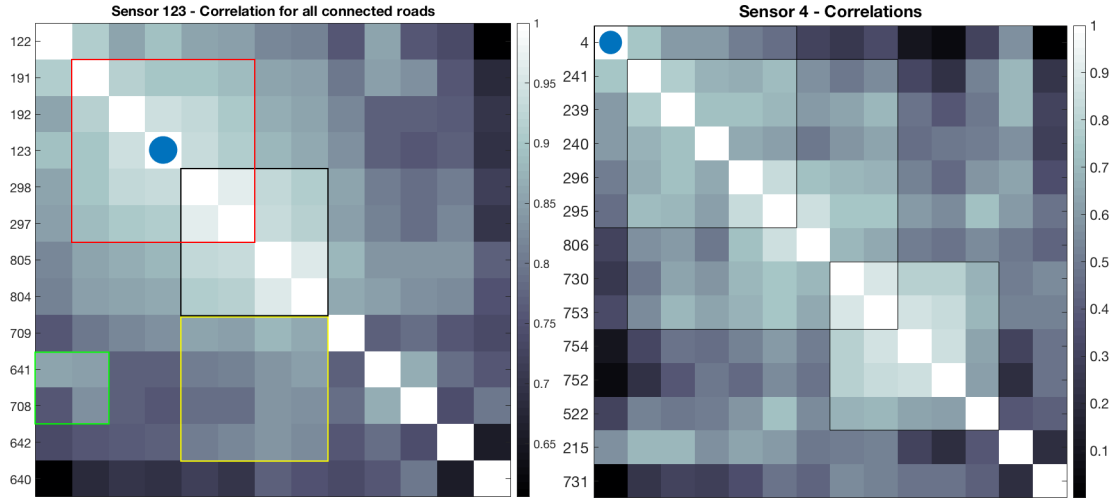
From Figure 4.8a I can observe that a cluster (red square) is formed with the perpendicular query road (123) at both ends, namely 191 and 192 at one end and 298 and 297 at the other end. While these consecutive sections are correlated with themselves. It is not known at which end of the road section the sensors are placed unfortunately for section 123 or other sections, hence I can not even attempt to make causality assumptions.

It can further be observed that in turn these last two form a *cluster* with



(a) Sensor 123 - directly connected roads **(b)** Sensor 4 - nearest via euclidean distance

Figure 4.7: Road sections are marked for either direction (unreliable).



(a) Sensor 123 **(b)** Sensor 4

Figure 4.8: Correlation matrix for two different road sections

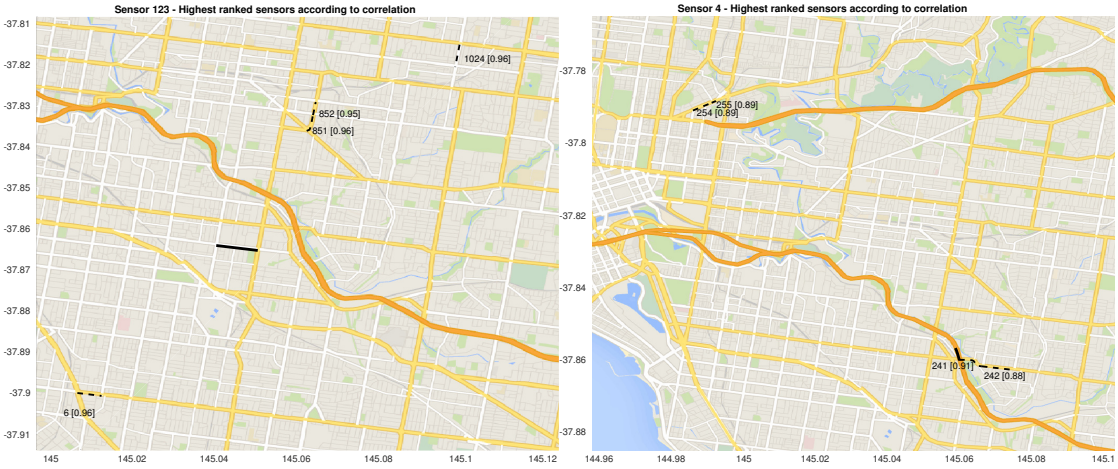
the roads on the opposite traffic (black square). The yellow circle depicts the correlations on the other side of the road and their correlation with the parallel side of the road. The green square shows that the opposite direction section (641) from the query road is more correlated with traffic from the top, bottom and left side. Upon even more careful inspection, these correlations reveal the behaviour of traffic despite the fact that the actual direction of traffic is now known – this is an undirected graph.

Conversely, in Figure 4.8b we can observe that there are *clusters* formed for the other road segments while segment 4 is only slightly correlated with section 241. Nevertheless, there are still weak correlations even at this relatively isolated location. Figure 4.7 shows that it is not necessary for a road section to have direct adjacent road sections. However, the traffic volume on this road is still influenced by the nearby on and off ramps. Hence, I plot in Figure 4.7b the closest road sections using the euclidean distance. Recent work optimize only for individual series forecasting and hence does not take proximity data into consideration towards making predictions [108], an observation also made by the authors. It is evident that there are dependencies between sensors in a traffic network, especially at intersections as I show in Figure 4.8.

Thus far we have observed proximity operators based on direct adjacent connections or euclidean distance. I use the correlation as a metric for selecting road sections as opposed to using the map coordinates for each sensor. In Figure 4.9 I show that the top most correlated roads for these two sensors are *not* necessarily in the immediate neighbourhood. Therefore, this is not a reliable means of selecting additional relevant data for the predictors, since it is quite unlikely that there are dependencies between road sections that are far apart. Perhaps a better way of selection is to compute the cross-correlation over a fixed window interval, which accounts for shifts in the traffic signal.

The correlations for all sensors in one area are shown in Figure 4.8. Clearly, the road section 123 has much more correlated traffic than the off highway ramp (section 4). This is due to the fact that the ramp has no actual connected roads and these roads were selected using the euclidean distance matrix. The implications are evident: there is valuable information carried within the neighbourhood of each query road section where predictions are to be made.

Previous research has shown that separating prediction on working and non-working (weekends) days can improve performance if independent predictors are deployed separately for weekends or each day of the week. Additionally, both a larger temporal context through a larger lag and including proximity data can increase prediction accuracy as was also shown in Chapter 3.



(a) Sensor 123 **(b)** Sensor 4

Figure 4.9: Ranked road sections by correlation. Query is solid black. The highest ranked are shown in dotted black. There is a large distance between the query and the highest correlated over the entire dataset.

4.4.6 Pair-wise correlations with query station as a function of time

I further investigate whether the interactions depicted in Figure 4.8 change during the day. In the following figure we can observe that these indeed change. There is an overall pattern, however most importantly there are deviations from the standard pattern such as the sudden spikes at 4 am for the blue sensor station and the spike at 6p.m. for the green sensor station.

4.5 Results and discussion

In this section I evaluate the network wide prediction performance for the two datasets and compare univariate methods to Topology Regularized Vector Autoregression (TRU-VAR) generalized to state of the art function approximators. In all experiments I only report the error of ex-ante point forecasts, in other words, no information that would not be available at the time of prediction is used in any of the experiments.

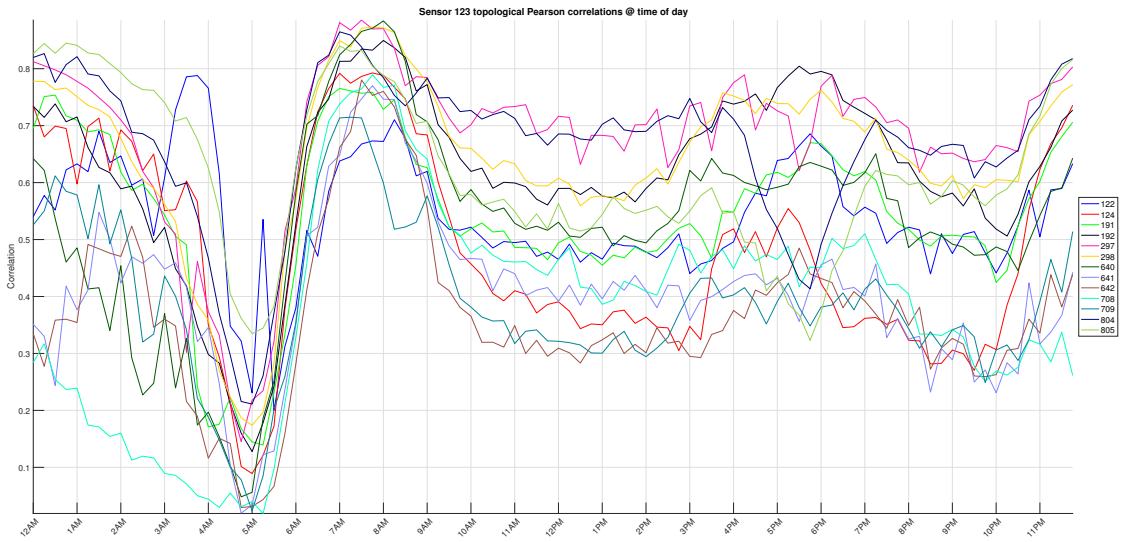


Figure 4.10: Sensor 123 correlation at time of day with neighbouring sensors

4.5.1 Experimental setup

For all experiments the data was split sequentially into 70% training 15% validation and 15% testing. Preparation of data is discussed in section 4.4. In the case of ARIMA, the models were fit on the training plus validation data. The regularization parameters and the fitting of the ARIMA (using the forecast package in R [79]) were found independently for each sensor station. The causal CNNs are trained using the gauss-newton approximation for Bayesian L_2 regularized backpropagation [53]. In the case of the LLS and SVR I use L-BFGS [131, ch. 9] for ridge and SPARSA [188] for LASSO regularization.

As a follow-up on the observations made in Figure 4.8a and Figure 4.8b I propose learning Topology Regularized Universal Vector Autoregression by inducing sparsity in the VAR model, in effect including only relevant data from the nearby roads to each autoregressor. However, not all road sections have direct connections. For the VicRoads dataset, I only use data from the directly connected roads, if the road section has direct connections. In Figure 4.7b the nearest roads for road section 4 are plotted, however there are no directly connected roads, a case where no additional data is added. In this case, the topological adjacency matrix can be refined according to other heuristics for selecting relevant roads, as discussed in section 4.3. PeMS is very sparse given the

area covered, I take the closest $K = 6$ roads as additional data, computed using the graph adjacency matrix from the map GPS coordinate of each sensor. I empirically selected K based on the out of sample (test set) mean RMSE using univariate OLS fitting.

I set a naive baseline using the CMSD as specified in section 4.4, which is the average specific to the time of day, sensor and day of the week. I furthermore compare these methods with ARIMA which is also an intuitive baseline. Note that I do not provide comparisons with VAR since this would be computationally prohibitive, especially for the VicRoads dataset where the input space would have almost 10k ($\Delta = 10 \times 1084$) dimensions, just for one vector. I do not perform direct comparisons with recent methods [95, 144, 190, 2, 54, 115, 192] since: 1) this would be computationally prohibitive – in the original articles, most authors do not perform network-wide experiments and the current dataset has 1000+ sensor stations; 2) these methods do not have the properties described in the introduction (Table 4.1 on 85); 3) I aim for a direct comparison over the univariate equivalent of the function approximator used within TRU-VAR.

Univariate models and TRU-VAR are compared via the average RMSE and MAPE for both datasets.

4.5.2 Choosing the lag order

I train a linear TRU-VAR over increasing lag windows $\Delta \in \{1, 2, 3, 5 \ldots, 31\}$ and plot the validation set RMSE and computation time, towards making an empirical choice for the lag value. A larger lag decreases prediction error, while putting a heavier load on processing time. As a tradeoff I set $\Delta = 10$ and use the same lag value for the VicRoads dataset.

Using this procedure I aim to get an estimate of the out of sample error. Using this lag value, the error on the test set is lower for models with the same lag value as opposed to the ARIMA forecasts where the lags were selected automatically for each timeseries based on the AIC. It is however likely that if optimizing the lag values in the same way for TRU-VAR, the error could be further lowered.

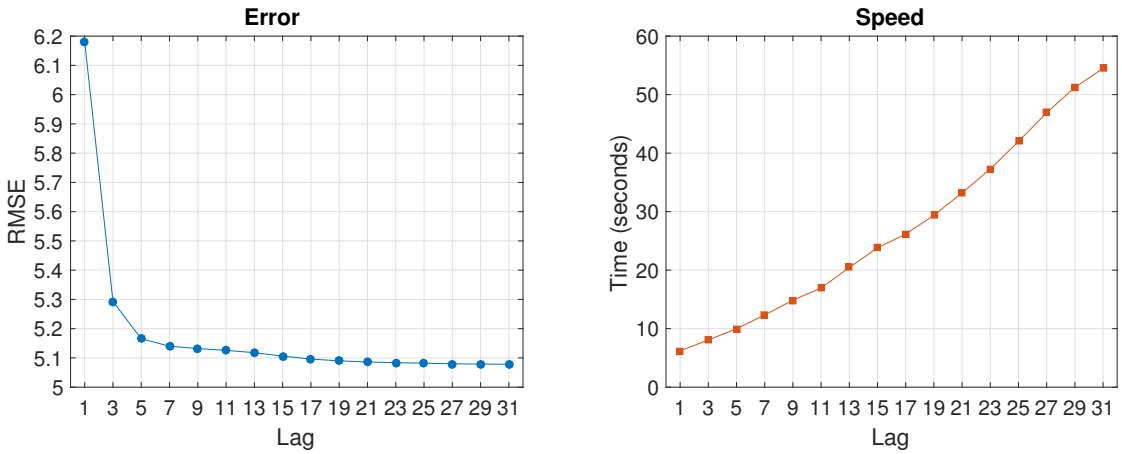


Figure 4.11: RMSE and prediction time as a function of lag (PeMS).

4.5.3 TRU-VAR vs. Univariate

It can be observed from Table 4.2 that TRU-VAR outperformed the univariate methods in all cases except for SVR-L_1.

Table 4.2: VicRoads dataset - Average **RMSE**
Topology regularized universal vector autoregression (TRU-VAR) outperforms univariate models for all f except SVR-L_1.

$f(x)$	OLS	LLS-L_1	LLS-L_2	SVR-L_1	SVR-L_2	CNN-L_2
Univariate	24.11	24.13	24.37	24.94	24.51	**22.09**
	$\pm$ 15.4	$\pm$ 15.4	$\pm$ 15.5	$\pm$ 15.9	$\pm$ 15.9	$\pm$ 15.6
TRU-VAR	**22.64**	**23.14**	**22.93**	26.68	**22.78**	**21.36**
	$\pm$ 15.6	$\pm$ 15.7	$\pm$ 15.4	$\pm$ 18.5	$\pm$ 15.1	$\pm$ 15.3
Baselines	CMSD: 35.29 $\pm$ 29.0			ARIMA: 24.32 $\pm$ 15.6		

For PeMS there are usually just two adjacent sensors (upstream and downstream) which can contribute to the traffic volume. I was unable to pinpoint the start and end location of the road section covered by the sensor station unlike VicRoads. Since only the GPS coordinates of stations themselves were available, I defined the TDAM based on the euclidean distance to the query sensor station. This usually resulted in including the sensor stations on the opposite direction of traffic. While this helps in the case of VicRoads, for freeways the

Table 4.3: VicRoads dataset - Average **MAPE** Topology regularized universal vector autoregression (TRU-VAR)
outperforms univariate models for all f except SVR-L_1.

$f(x)$	OLS	LLS-L_1	LLS-L_2	SVR-L_1	SVR-L_2	CNN-L_2
Univariate	26.94	27.14	28.42	28.22	27.56	**21.27**
	$\pm$ 12.4	$\pm$ 12.6	$\pm$ 14.2	$\pm$ 14.6	$\pm$ 14.6	$\pm$ 8.5
TRU-VAR	**22.96**	**24.53**	**24.16**	31.91	**24.28**	**21.03**
	$\pm$ 9.9	$\pm$ 13.8	$\pm$ 12.5	$\pm$ 26.3	$\pm$ 13.7	$\pm$ 11.7
Baselines	CMSD: 32.86 $\pm$ 53.4			ARIMA: 27.91 $\pm$ 13.4		

traffic is strictly separated between traffic directions, hence it is not relevant and adds unnecessary complexity. However, it is interesting that the prediction error was lowered by defining the adjacency matrix based on the K roads furtherest apart from the prediction location for the OLS and Causal CNN. The results are shown in Table 4.4 in the last row. From the same table it can be seen that for higher temporal resolutions such as in the case of PeMS, the lowest errors are recorded via OLS and Causal CNN while ARIMA, LLS and SVR show higher error.

Table 4.4: PeMS dataset - Average **RMSE**
Topology regularized universal vector autoregression (TRU-VAR)
outperforms univariate models for all f except LLS-L_1.

$f(x)$	OLS	LLS-L_1	LLS-L_2	SVR-L_1	SVR-L_2	CNN-L_2
Univariate	4.61	4.64	4.97	5.20	4.72	**4.55**
	$\pm$ 2.1	$\pm$ 2.0	$\pm$ 2.4	$\pm$ 2.6	$\pm$ 2.0	$\pm$ 2.0
TRU-VAR	**4.53**	4.64	**4.92**	**5.03**	**4.70**	**4.45**
	$\pm$ 2.0	$\pm$ 2.1	$\pm$ 2.5	$\pm$ 2.7	$\pm$ 2.0	$\pm$ 1.9
Baselines	CMSD: 5.16 $\pm$ 3.1			ARIMA: 4.67 $\pm$ 1.7		

Table 4.5: PeMS dataset - Average **MAPE**
Topology regularized universal vector autoregression (TRU-VAR) outperforms univariate models for all f except LLS-L_1.

$f(x)$	OLS	LLS-L_1	LLS-L_2	SVR-L_1	SVR-L_2	CNN-L_2
Univariate	37.77	38.37	45.12	42.80	38.53	**37.48**
	$\pm$ 9.4	$\pm$ 11.0	$\pm$ 15.9	$\pm$ 20.4	$\pm$ 13.5	$\pm$ 9.8
TRU-VAR	**36.95**	38.37	**41.59**	**39.74**	**37.69**	**37.42**
	$\pm$ 10.3	$\pm$ 12.5	$\pm$ 14.7	$\pm$ 14.5	$\pm$ 12.3	$\pm$ 12.6
Baselines	CMSD: 41.9 $\pm$ 19.2			ARIMA: 38.09 $\pm$ 15.76		

4.5.4 Long-term forecasting: increasing the prediction horizon

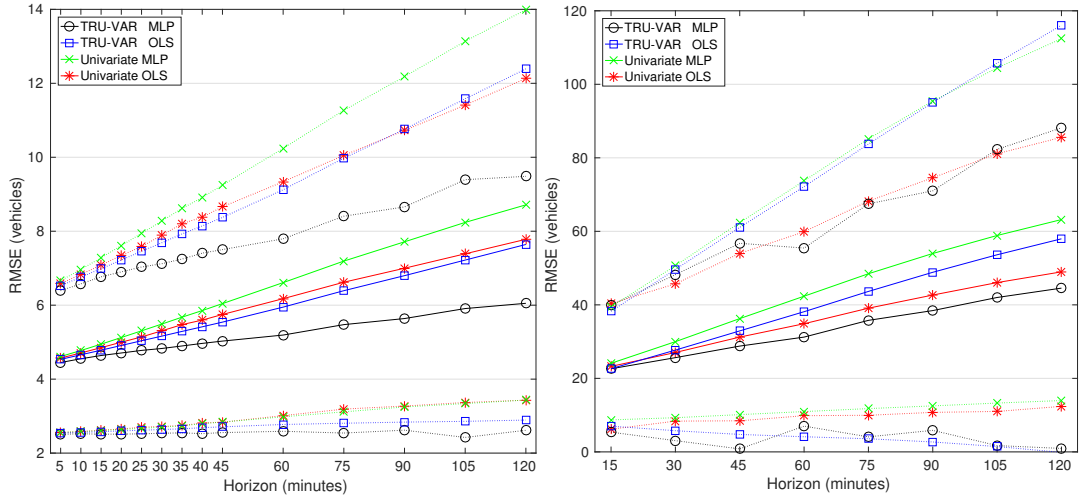
In the previous section I showed that TRU-VAR outperforms univariate methods across different machine learning function approximation models. I now ask if this holds for larger prediction horizons for to up to two hours. Consequently, I compare the prediction performance of TRU-VAR and univariate models with the two best performing models from the previous section (OLS and CNN). The experiment is identical to the one in the previous section, with the exception that now, instead of predicting at the immediate step in the future (e.g. $h = 1$) I set the target variable to be further in time. In other words, methods are identical data split is identical, input data is the same while the target variable changes.

Therefore, in the following section I show how the overall network error rises as the prediction horizon is increased to up to two hours. I now refer the reader to Equation 4.1 on page 95 for the definition of the prediction horizon. The horizon temporal resolution for $t + h$ where $h = 1$ was initially 5 minutes for PeMS and 15 minutes for VicRoads which correspond to the temporal resolution of the dataset.

Instead of predicting the volume of traffic at $t + 1$ (for all series) I instead train and subsequently evaluate the prediction error at horizons for up to two hours. For PeMS, which has a resolution of 5 minutes per observation I plot the error for eight 5-minute distanced horizons and then add 15-minute hori-

zons up to two hours (I skip a few horizons) – this is visible in Figure 4.12a. For VicRoads the resolution is 15 minutes thus I only plot 8 forecasts to reach the two hour goal (Figure 4.12b). Therefore, to evaluate all horizons for up to two hours I require to fit $h \in \{1, 2, \ldots, 8\}$ models for VicRoads and $h \in \{1, 2, \ldots, 24\}$ for PeMS. However, for PeMS I skip a few evaluations and take $h \in \{1, 2, 3, \ldots, 8\} \cup \{9, 12, \ldots, 24\}$.

Results are displayed in Figure 4.12 where the network-mean RMSE is depicted with solid lines and the standard deviation with dotted lines as an indication of the network-spread of the error. It is evident that it is more challenging to make predictions further in time, and it can be seen that the error increases linearly as the prediction horizon is increased.



(a) PeMS **(b)** VicRoads

Figure 4.12: Behaviour of error when prediction horizon is increased. μ RMSE with solid lines and spread of $\mu \pm \sigma$ RMSE with dotted lines. Lower is better, error increases linearly with the prediction horizon.

From both figures it can be observed that TRU-VAR CNN predicts with the lowest error even as the forecasting horizon is increased, while the Univariate CNN (one simple neural network per timeseries) is the worst for both datasets. As the prediction horizon is moved further ahead in time not only the mean RMSE increases, but the spread of the error over the network also increases. For PeMS the error increases by approximately 36% when the horizon is moved

from five minutes to two hours (for TRU-VAR CNN) while for VicRoads, the error increases by 97%. This is to be expected for a much larger network, with a more complex topology, a greater variety of road types and a lower temporal resolution.

Overall, the error spread appears to be much more stable for PeMS and the OLS approximators. This is for two reasons: 1) VicRoads has 10 times more sensor stations, which can cause a much larger error spread; 2) for the causal CNN experiments the neural network was trained using first order gradient methods which was faster, however less stable than when using Bayesian regularization as in the previous experiments in Table 4.2.

4.6 Conclusions and future work

In the current chapter I defined necessary properties for large scale network wide traffic forecasting in the context of growing urban road networks and (semi)autonomous vehicles. I performed a broad review of the literature and after drawing conclusions, I discussed data quality, preprocessing and provided suggestions for defining a topology-designed adjacency matrix (TDAM).

I consequently proposed topology-regularized universal vector autoregression (TRU-VAR) and showed that causal CNNs are the best performing model for the current application. I compared the network-wide prediction error of the univariate and TRU-VAR prediction models over two quantitatively and qualitatively different datasets. TRU-VAR outperforms the CMSD baseline and ARIMA in all cases and the regularized univariate models in almost all cases. For VicRoads, which has high spatial but relatively lower temporal resolution, TRU-VAR outperformed the univariate method in all cases except for SVR-L_1. For the PeMS dataset, TRU-VAR showed lower error in all cases except for LLS-L_1. From the prediction horizon experiments (Figure 4.12) it can be concluded that the TRU-VAR causal CNN approximator has the lowest error even when the forecast horizon is increased up to two hours, for both datasets. Therefore, very simple CNNs can be used as effective models for multi-task learning for time-series data.

PeMS was easier to predict and the RMSE is lower than for VicRoads. I

would like to remind the reader that approximately half of the sensor stations in the VicRoads dataset can have more than 50% data missing. I did not remove these from the model. This evidently increases the overall error. At the same time, the PeMS dataset has higher temporal resolution (5 vs 15 minutes) however it is much smaller and has 10 times less sensor stations. With continuous state-space models and accounting for the spatial sparsity (large distance between sensor stations on highways causes shifts in the signals) the error could be further decreased.

The TDAM is a key component of the multi-task learning framework system since it has a great impact on prediction accuracy and should be tailored to the type of road network using domain knowledge. Therefore, the weak points of the method also reside in its strengths, namely: the design of the topological adjacency matrix is very important and is subject to domain knowledge; the model customization flexibility allows for a large search space of possibilities which can be overwhelming to fine-tune, if required. When designing the adjacency matrix for the VicRoads dataset, I could have opted to rank the most correlated connected ones and select the top K ranked ones, in this way having an equal number of additional streams for each prediction point. For the roads without direct connections, this would require computing the distance to the closest roads based on the euclidean distance and performing a correlation ranking. This could further increase accuracy since for some sensors (for example highway off ramps) there are no directly connected roads, hence simple regression is performed. An alternative would have been to select second order connections for the stations with a low graph node degree, such that of highway off ramps. I leave this for future work, also aiming to investigate continuous state space models, multi-metric data and other multi-task learning architectures. Moreover, I also aim to show that including temporal features and adding multi-metric data (such as vehicle-crowdsourced data) can further increase prediction performance.

Finally, I would like to point out that the method can be trained online, with very simple and efficient CNNs, has a low complexity due to the topological constraints, is non-static and robust towards changes (data sources or structure) thus scales well, is efficient and easy to redeploy. Given that the error increases

linearly within reasonable bounds (Figure 4.12) for up to two hours, the method may be useful in other contexts and applications other than road traffic, such as natural disaster prevention (e.g. river flood forecasting), telecommunications (e.g. antenna load forecasts), networking (e.g. forecasts for routing), finance to name a few. In general, the method can be applied to any type of dataset consisting of multivariate timeseries, where the constraints are known a priori or can be inferred from the data to construct the topology matrix.

Chapter 5

SynthNet: Learning synthesizers end-to-end

This chapter is based on the following peer-reviewed publication:

F. Schimbinschi, C. Walder, S.M. Erfani, J. Bailey, "*SynthNet: Learning synthesisers end-to-end*" under review **International Conference on Learning Representations**, (ICLR) 2019.

Learning synthesizers and generating music in the raw audio domain is a challenging task. I investigate the learned representations of convolutional autoregressive generative models. Consequently, I show that mappings between musical notes and the harmonic style (instrument timbre) can be learned based on the raw audio music recording and the musical score (in binary piano roll format). The proposed architecture, SynthNet uses minimal training data (9 minutes), is substantially better in quality and converges 6 times faster than the baselines. The quality of the generated waveforms (generation accuracy) is sufficiently high that they are almost identical to the ground truth. Therefore, I am able to directly measure generation error during training, based on the RMSE of the Constant-Q transform. Mean opinion scores are also provided. I validate this work using 7 distinct harmonic styles and also provide visualizations and links to all generated audio.

5.1 Introduction

WaveNets [170] have revolutionized text to speech by producing realistic human voices. Even though the generated speech sounds natural, upon a closer inspection the waveforms are different to genuine recordings. As a natural progression, I propose a WaveNet derivative called SynthNet which can learn and render (in a controlled way) the complex harmonics in the audio training data, to a high level of fidelity. While vision is well established, there is little understanding over what audio generative models are learning. Towards enabling similar progress, I give a few insights into the learned representations of WaveNets, upon which I build the model.

WaveNets were trained using raw audio waveforms aligned with linguistic features. The current work takes a similar approach to learning music synthesizers and trains the model based on the raw audio waveforms of entire songs and their symbolic representation of the melody. This is more challenging than speech due to the following differences: 1) in musical compositions multiple notes can be played at the same time, while words are spoken one at a time; 2) the timbre of a musical instrument is arguably more complex than speech; 3) semantically, utterances in music can span over a longer time.

[170] showed that WaveNets can generate new piano compositions based on raw audio. Recently, this work was extended by [43], delivering a higher consistency in compositional styling. Closer to the current work, [50] describe a method for learning synthesizers based on individually labelled note-waveforms. This is a laborious task and is impractical for creating synthesizers from real instruments. The currently proposed method bypasses this problem since it can directly use audio recordings of an artist playing a given song, on the target instrument.

SynthNet can learn representations of the timbre of a musical instrument more accurately and efficiently via the dilated blocks through depthwise separable convolutions. I show that it is enough to condition only the first input layer, where a joint embedding between notes and the corresponding fundamental frequencies is learned. I remove the skip connections and instead add an additional loss for the conditioning signal. I also use an embedding layer for

the audio input and use SeLU [94] activations in the final block.

The benchmarks against the WaveNet [170] and DeepVoice [9] architectures show that the current method trains faster and produces high quality audio. After training, SynthNet can generate new audio waveforms in the target harmonic style, based on a given song which was not seen at training time. While I focus on music, SynthNet can be applied to other domains as well. Convolutional autoregressive generative models have applications in a broad range of domains with streaming data and have recently seen an increased interest [66, 16]. Furthermore, the achievements of this work have have a broader implication in adversarial learning for time-series domains since the generated data (i.e. fake / artificial data) is much closer to the real data.

Contributions are as follows: 1) I show that musical instrument synthesizers can be learned end-to-end based on raw audio and a binary note representation, with minimal training data. Multiple instruments can be learned by a single model. **2)** I give insights into the representations learned by dilated causal convolutional blocks and consequently propose SynthNet, which provides substantial improvements in quality and training time compared to previous work. Indeed, I demonstrate (Figure 5.8) that the generated audio is practically identical to the ground truth. **3)** The benchmarks against existing architectures contains an extensive set of experiments spanning over three sets of hyperparameters, where I control for receptive field size. I show that the RMSE of the Constant-Q Transform (RMSE-CQT) is highly correlated with the mean opinion score (MOS). **4)** I find that reducing quantization error via dithering is a critical preprocessing step towards generating the correct melody and learning the correct pitch to fundamental frequency mapping.

5.2 Related work

In music, style can be defined as the holistic combination of the melodic, rhythmic and harmonic components of a particular piece. The delay and sustain variation between notes determines the *rhythmic style*. The latter can vary over genres (e.g. Jazz vs Classical) or composers. Timbre or *harmonic style* can be defined as the short term (attack) and steady state (sustained frequency dis-

tribution) acoustical properties of a musical instrument [150]. The focus is on learning the *harmonic style*, while controlling the (given) melodic content and avoiding any rhythmic variations.

The research on content creation is plentiful. For an in depth survey of deep learning methods for music generation I point the reader to the work of [20]. Generative autoregressive models were used in [170, 119] to generate new random content with similar harmonics and stylistic variations in melody and rhythm. Recently, the work of [170] was extended by [43] where the quality is improved and the artificial piano compositions are more realistic. I have found piano to be one of the easier instruments to learn. [48] introduce WaveGANs for generating music with rhythmic and melodic variations.

Closer to the current work, [50] propose WaveNet Autoencoders for learning and merging the harmonic properties of instrument synthesizers. The major difference with this work is that I was able to learn harmonic styles from entire songs (a mapped sequence of notes to the corresponding waveform), while their method requires individually labelled notes (NSynth dataset). With my method the overhead of learning a new instrument is greatly reduced. Moreover, SynthNet requires minimal data and does not use note velocity information.

Based on the architecture proposed by [50], and taking a domain adaptation approach, [126] condition the generation process based on raw audio. An encoder is used to learn note mappings from a source audio timbre to a target audio timbre. The approach can be more error prone than ours, since it implies the intermediary step of correctly decoding the right notes from raw audio. This can significantly decrease the generation quality. Interestingly, [126] play symphonic orchestras from a single instrument audio. However, there is no control over which instrument plays what. Conversely, I use individual scores for each instrument, which gives the user more control. This is how artists usually compose music.

5.3 End-to-end synthesizer learning

[170] and [9] have shown that generative convolutional networks are effective at learning human voice from raw audio. This has advanced the state of the art

in text to speech (TTS). Here, I further explore the possibilities of these architectures by benchmarking them in the creative domain – learning music synthesizers. There are considerable differences between the human voice and musical instruments. Firstly, the harmonic complexity of musical instruments is higher than the human voice. Second, even for single instrument music, multiple notes can be played at the same time. This is not true for speech, where only one sound utterance is produced at a time. Lastly, the melodic and rhythmic components in a musical piece span a larger temporal context than a series of phonemes as part of speech. Therefore, the music domain is much more challenging.

5.3.1 Baseline architectures

The starting point is the model proposed by [170] with the subsequent refinements in [9]. I refer the reader to the these articles for further details. The data consists of triplets $\{(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N, \boldsymbol{z}_S)\}$ over N songs and S styles, where $\boldsymbol{x}_i$ is the 256-valued encoded waveform, $\boldsymbol{y}_i$ is the 128-valued binary encoded MIDI and $\boldsymbol{z}_s \in \{1, 2, \ldots, S\}$ is the one-hot encoded style label. Each audio sample x_t is conditioned on the audio samples at all previous time steps $\boldsymbol{x}_{<t} = \{x_{t-1}, x_{t-2}, \ldots, x_1\}$, all previous binary MIDI samples and the global conditioning vector. The joint probability of a waveform $\boldsymbol{x} = \{x_1, \ldots, x_T\}$ is factorized as follows:

$$p(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}) = \prod_{t=1}^{T} p(x_t \mid \boldsymbol{x}_{<t}, \boldsymbol{y}_{<t}, \boldsymbol{z}). \tag{5.1}$$

The hidden state before the residual connection in dilation block ℓ is

$$\boldsymbol{h}^{\ell} = \tau\Big(\boldsymbol{W}_f^{\ell} * \boldsymbol{x}^{\ell-1} + \boldsymbol{V}_f^{\ell} * \boldsymbol{y}^{\ell-1} + \boldsymbol{U}_f^{\ell} \cdot \boldsymbol{z}\Big) \odot \sigma\Big(\boldsymbol{W}_g^{\ell} * \boldsymbol{x}^{\ell-1} + \boldsymbol{V}_g^{\ell} * \boldsymbol{y}^{\ell-1} + \boldsymbol{U}_g^{\ell} \cdot \boldsymbol{z}\Big), \tag{5.2}$$

while the output of every dilation block, after the residual connection is

$$\boldsymbol{x}^{\ell} = \boldsymbol{x}^{\ell-1} + \boldsymbol{W}_r^{\ell} \cdot \boldsymbol{h}^{\ell}, \tag{5.3}$$

where τ and σ are respectively the tanh and sigmoid activation functions, ℓ is the layer index, f indicates the filter weights, g the gate weights, r the residual weights and $\boldsymbol{W}$, $\boldsymbol{V}$ and $\boldsymbol{U}$ are the learned parameters for the main, local conditioning and global conditioning signals respectively. The f and g convolutions are computed in parallel as a single operation [9]. All convolutions have a filter width of F. The convolutions with $\boldsymbol{W}^\ell$ and $\boldsymbol{V}^\ell$ are dilated.

To locally condition the audio signal, [170] first upsample the $\boldsymbol{y}$ time series to the same resolution as the audio signal (obtaining $\boldsymbol{y}^\ell$) using a transposed convolutional network, while [9] use a bidirectional RNN. In this case, the binary midi vector already has the same resolution.

I use an initial causal convolution layer (Equation 5.4) that only projects the dimensionality of the signal from 128 channels to the number of residual channels. The first input layers are causal convolutions with parameters $\boldsymbol{W}^0$ and $\boldsymbol{V}^0$ for the waveform and respectively the piano roll:

$$\boldsymbol{y}^\ell = \boldsymbol{V}^0 * \boldsymbol{y}, \ \forall \ell \tag{5.4}$$

$$\boldsymbol{x}^0 = \boldsymbol{W}^0 * \boldsymbol{x}. \tag{5.5}$$

All other architecture details are kept identical to the ones presented in [170] and [9] as best as I could determine. The differences between the two architectures and SynthNet are summarized in Table 5.1. I compare the performance and quality of these two baselines against SynthNet initially in Table 5.3 over three sets of hyperparameters (Table 5.2). For the best resulting models I perform MOS listening tests, shown in Table 5.5. Preliminary results for global conditioning experiments are also provided in Table 5.4.

Table 5.1: Differences between the two baseline architectures and SynthNet.

	Input			Dilated conv	Skip		Final block
	Channels	Type	Activation	Separable	Connection	1x1 Conv	Activation
WaveNet	1 Scalar	Conv	None	No	Yes	No	ReLU
DeepVoice	256 1-hot	Conv	Tanh	No	Yes	Yes	ReLU
SynthNet	1 Scalar	Embed	Tanh	Yes	No	No	SeLU

5.3.2 Gram matrix projections

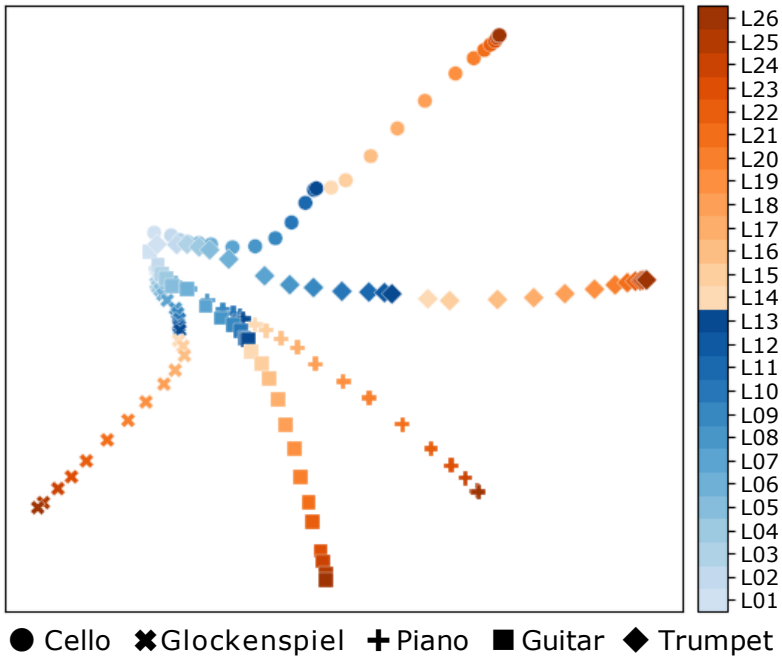


Figure 5.1: Gram matrix projection from Eq. 5.3 Layers in colour, shapes are styles (timbre).

I perform a set of initial experiments to gain more insight towards the learned representations. I use Gram matrices to extract statistics since these have been previously used for artistic style transfer [56]. After training, the validation data is fed through five locally conditioned networks, each trained with a distinct harmonic style. The data has identical *melodic* content but has different *harmonic* content (i.e. same song, different instruments). The Gram matrices are extracted from the outputs of each dilated block (Equation 5.3) for each network - timbre.

These are flattened and projected onto 2D, simultaneously over all layers and styles via T-SNE [116]. The results presented in Figure 5.1 show that the extracted statistics separate further as the layer index increases. A broad interpretation is that the initial layers extract low-level generic audio features, these being common to all waveforms. However, since this is a controlled experiment, I can be more specific. The timbre of a musical instrument is characterized by a

specific set of resonating frequencies on top of the fundamental frequency (pure sine wave). Typically one identifies individual notes based on their fundamental, or lowest prominent frequency. These depend on the physics of the musical instrument and effects generated, for example, by the environment. Since the sequence of notes is identical and the harmonic styles differ, I conjectured that Figure 5.1 could imply a frequency-layer correspondence. While the latter statement might be loose, the lower layers' statistics are nevertheless much closer due the increased similarity with the fundamental frequency.

5.3.3 SynthNet architecture

Figure 5.1 provides indicative results from many experiments. Additional Gram matrix projections are provided in Figure 5.10. I hypothesize that the skip connections are superfluous and the conditioning of the first input layer should suffice to drive the melodic component. I also hypothesize that the first audio input layer learns an embedding corresponding to the fundamental pitches. Then, I aim to learn mapping from the symbolic representation (binary midi code) to the pitch embeddings (Equation 5.8). Therefore, in SynthNet there are no parameters learned in each dilation block for local conditioning and the hidden activation with global conditioning (omitted in Figure 5.2) becomes

$$\boldsymbol{h}^{\ell} = \tau\Big(W_f^{\ell} * \boldsymbol{x}^{\ell-1} + U_f^{\ell} \cdot \boldsymbol{z}\Big) \odot \sigma\Big(W_g^{\ell} * \boldsymbol{x}^{\ell-1} + U_g^{\ell} \cdot \boldsymbol{z}\Big). \tag{5.6}$$

The input to the dilated blocks is the sum of the embedding codes and the autoencoder latent codes:

$$\boldsymbol{y}^h = \tau(V^0 \cdot \boldsymbol{y}) \tag{5.7}$$

$$\boldsymbol{x}^0 = \tau(W^0 \cdot \boldsymbol{x}) + \tau(V^h \cdot \boldsymbol{y}^h) \tag{5.8}$$

$$\hat{\boldsymbol{y}} = V^{out} \cdot \tau(V^h \cdot \boldsymbol{y}^h). \tag{5.9}$$

As it can be seen in Figure 5.2 there are no skip connections and Equation 5.4 no longer applies. I also found that using SeLU activations [94] in the last layers improves generation stability and quality. Other normalization strategies could

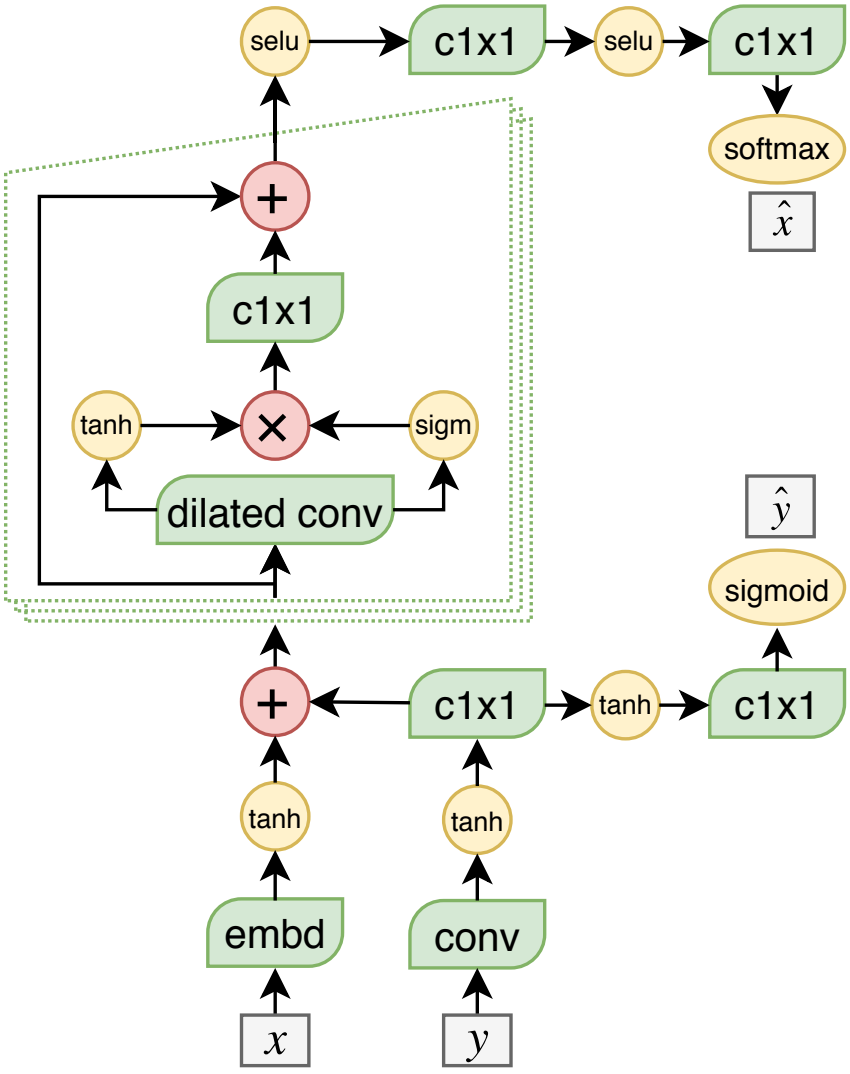


Figure 5.2: SynthNet (also see Table 5.1) with a multi-label cross-entropy loss for binary midi.

have been used, I found SeLU to work well. In addition, I further increase sparsity by changing the dilated convolution in Equation 5.6 with a dilated depthwise separable convolution. Separable convolutions perform a channel-wise spatial convolution that is followed by a 1×1 convolution. In this case each input channel is convolved with its own set of filters. Depthwise separable convolutions have been successfully used in mobile and embedded applications [75] and in the Xception architecture [33]. As I show in Table 5.4 and Table 5.5, the parsimonious approach works very well since it reduces the complexity of the architecture and speeds up training.

In training, SynthNet models the midi data $\boldsymbol{y}$ in an auto-regressive fashion which is similar to the way audio data is modelled, but with a simplified architecture (see Figure 5.2). In principle, this allows the model to jointly generate both audio and midi. In practice, the midi part of the model is too simple to generate interesting results in this way. Nonetheless, I found it beneficial to retain the midi loss term during training, which it turns out, tends to act as a useful regularizer — I conjecture by forcing basic midi features to be extracted. In summary, in contrast with Equation 5.1 I optimize the joint $\log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{z})$, so that

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{|\boldsymbol{x}|=256} x_j^i \log \hat{x}_j^i + \sum_{j=1}^{|\boldsymbol{y}|=128}\left(y_j^i \log \hat{y}_j^i + (1-y_j^i)\log(1-\hat{y}_j^i)\right)\right].$$

5.4 Experiments

I compare exact replicas of the architectures described in [170, 9] with the proposed architecture SynthNet. I train the networks to learn the harmonic audio style (here instrument timbre) using *raw audio waveforms*. The network is conditioned locally with a 128 binary vector indicating note on-off, extracted from the *midi files*. The latter describes the melodic content. For the purpose of validating the hypothesis, I decided to eliminate extra possible sources of error and manually upsampled the midi files using linear interpolation. For the results in Table 5.4 the network is also conditioned globally with a one-hot vector

which designates the style (instrument) identity. Hence, multiple instrument synthesizers are learned in a single model. For the hyperparameter search experiments (Table 5.3) and the final MOS results (Table 5.5) I train one network for each style, since it is faster.

I use the Adam [90] optimization algorithm with a batch size of 1, a learning rate of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ with a weight decay of 10^{-5}. I find that for most instruments 100-150 epochs is enough for generating high quality audio, however I keep training up to 200 epochs to observe any unexpected behaviour or overfitting. All networks are trained on Tesla P100-SXM2 GPUs with 16GB of memory.

5.4.1 Audio and midi preprocessing

Audio is usually digitally encoded and distributed at 16 bits with a sampling rate of 44.1KHz where a continuous analog waveform is recorded as a succession of discrete amplitude values. The range of the amplitude values represents the bit depth.

The sampling rate (Figure 5.3) determines how many samples are stored in one second of audio recording and can also be thought of as the granularity of the data. The higher the sampling rate, the higher the frequencies that can be recorded accurately. Here, the audio sampling rate is reduced from 44.1KHz to 16KHz (16000 samples a second) which leads to satisfactory audio quality, even though some of the higher frequencies are not going to be reproduced well.

Bit depth determines the dynamic range of the audio signal which is 96 dB for 16-bit audio. In the studio, audio recordings are usually made at 24-bit which has a range of 144 dB - however current digital audio converter technology is not close to the upper limit. After mastering, the audio is usually quantized to 16-bit for commercial purposes. It is often the case that the audio is *dithered* before the quantization process is performed, in order to reduce the quantization error.

The concept of dithering is perhaps best exemplified with images. In Figure 5.4 the number of possible values for one pixel is reduced from 8 bit (2^8) in the left image, to 1 bit in the image on the right by truncating. Noise at the 50%

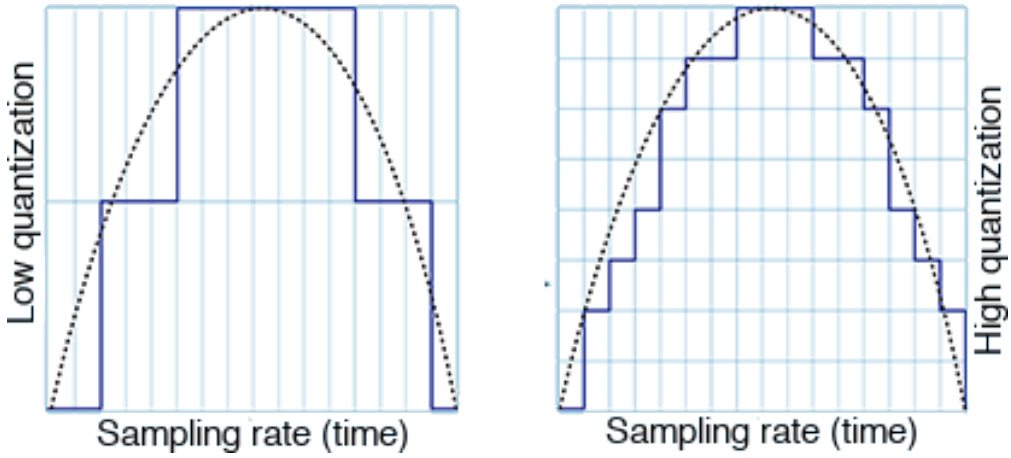


Figure 5.3: A signal (dotted line) can be approximated using a lower (left) or higher (right) quantization. Here, the sampling rate (horizontal) is the same in both cases.

grey level is added before reducing the bit depth. This captures the information in the original image at a lower detail, with relative grey levels kept. This statistically explains the original image since the 15% grey areas have a 15% chance of being black. The process of dithering for audio is exemplified in Figure 5.5.

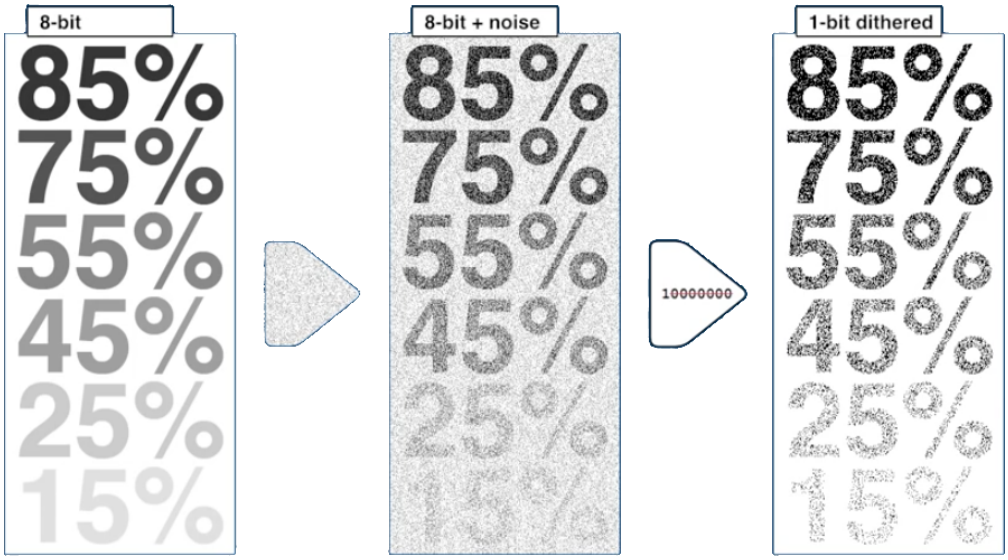


Figure 5.4: An 8 bit image is quantized to 1 bit. The process involves first adding noise, before reducing the precision in every pixel. This statistically captures the information in the signal, however at a lower detail fidelity.

In the current work, the audio is quantized using μ-Law companding and

subsequently encoded to a one-hot 256 valued vector. μ-Law companding only changes the bit depth of the audio from $2^16 = 65536$ to $2^8 = 256$ which makes training possible using the cross-entropy loss, where the values are further one-hot encoded into a vector. The inputs can still be scalar (but discretized) or a one-hot vector.

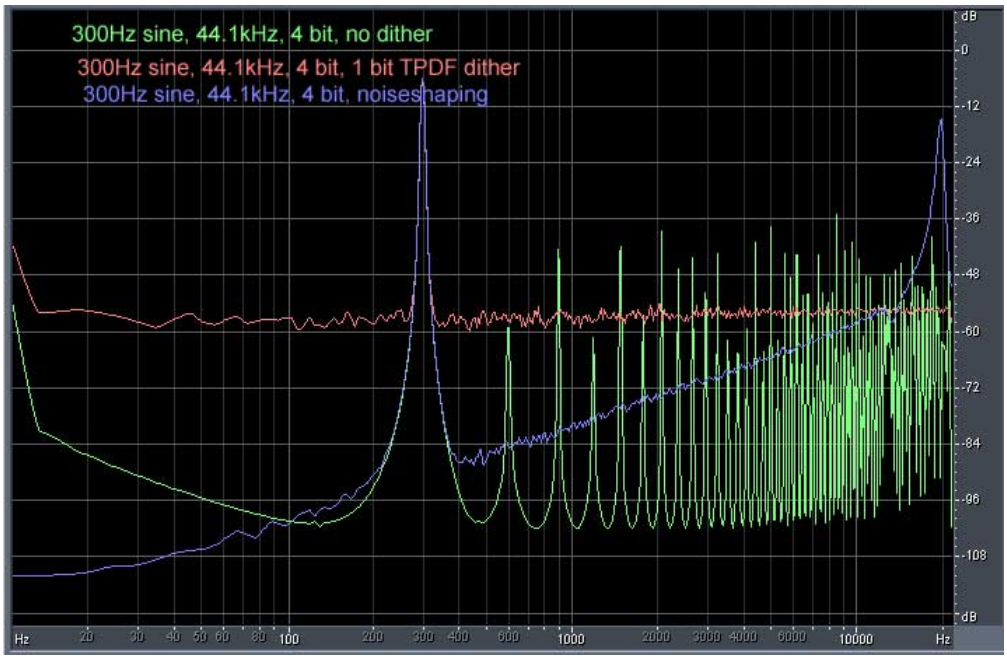


Figure 5.5: An abrupt quantization of an audio signal produces correlated noise patterns (green - right). Dithering implies adding noise (red signal) before the signal is quantized in order to mask the noise. In more advanced cases a noise shaping filter (blue) can be used.

Midi to piano roll A midi file encodes the time duration, initial velocity and pitch of a note. Here, for training, only the pitch information is kept which reduces the midi to a piano roll (Figure 5.6) - which is defined by $\boldsymbol{y}$. At one particular frame, one or more notes can be on - the values in the vector for one frame are not mutually exclusive.

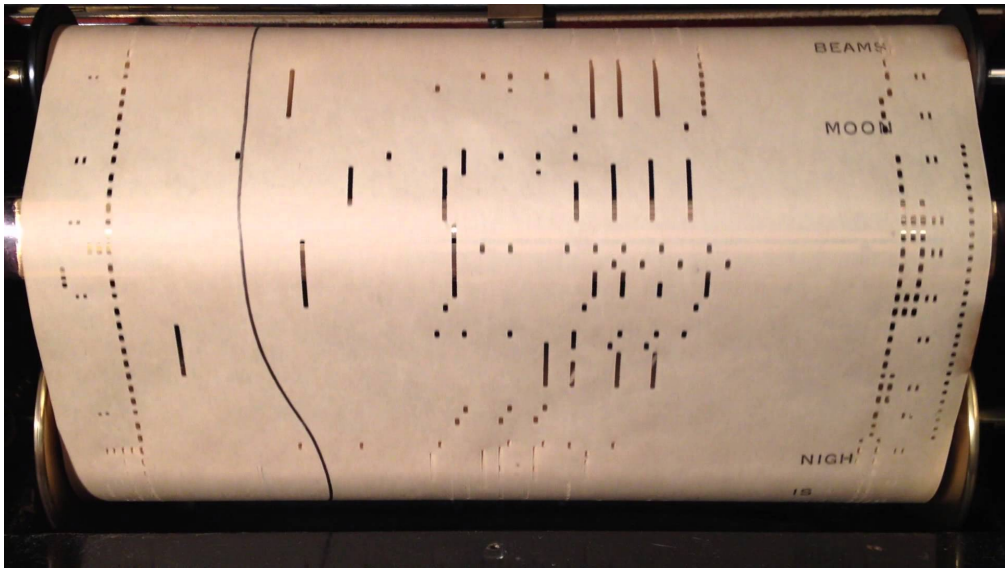


Figure 5.6: A vintage piano roll used to describe note on-off times.

5.4.2 Synthetic registered audio

The dataset is generated using the freely available Timidity++[1] software synthesizer. For training parts 2 to 6 from Bach's Cello Suite No. 1 in G major (BWV 1007) are selected. I found that this was enough to learn the mapping from midi to audio and to capture the harmonic properties of the musical instruments. From this suite, the Prelude (since it is most commonly known) is not seen during training and is used for measuring the validation loss and for conditioning the generated audio. The rest of the pieces (5) are used for training.

After synthesizing the audio, there are approximately 12 minutes of audio for each harmonic style, out of which 9 minutes (75%) training data and 3 minutes (25%) of validation data. I experiment with $S = 7$ harmonic styles which were selected to be as different as possible. Each style corresponds to a specific preset from the 'Fluid-R3-GM' sound font. These are (preset number - instrument): S01 - Bright Yamaha Grand, S09 - Glockenspiel, S24 - Nylon String Guitar, S42 - Cello, S56 - Trumpet, S75 - Pan Flute and S80 - Square Lead.

For training, the single channel waveforms are sampled at 16kHz and the bit-depth is reduced to 8 bit via mu-law encoding. Before reducing the audio

[1] http://timidity.sourceforge.net/

bit depth, *the waveforms are dithered* using a triangular noise distribution with limits $(-0.009, 0.009)$ and mode 0, which reduces perceptual noise but more importantly keeps the quantization noise out of the signal frequencies. I have found this critical for the learning process.

Without dithering there are melodic discontinuities and clipping errors in the generated waveforms. The latter errors are most likely due to notes getting mapped to the wrong set of frequencies (artefacts appear due to the quantization error). From all harmonic styles, the added white noise due to dithering is most noticeable for Glockenspiel, Cello and Pan Flute. The midi is upsampled to 16kHz to match the audio sampling rate via linear interpolation. Each frame contains a 128 valued vector which designates note on-off times for each note (piano roll).

5.4.3 Measuring audio generation quality

Quantifying the performance of generative models is not a trivial task. Similarly to [170, 9] I have found that once the training and validation losses go beyond a certain lower threshold, the quality improves. However, the losses are only informative towards convergence and overfitting (Figure 5.7) - they are not sensitive enough to accurately quantify the quality of the generated audio. This is critical for ablation studies where precision is important. [168] argue that generative models should be evaluated directly. Then, the first option is the mean opinion score (MOS) via direct listening tests. This can be impractical, slowing down the hyperparameter selection procedure. MOS ratings for the best found models are given in Table 5.5.

Instead, I propose to measure the root mean squared error (RMSE) of the Constant-Q Transform (RMSE-CQT) between the generated audio and the ground truth waveform (Figure 5.7, lower plots). Similarly to the Fourier transform, the CQT [21] is built on a bank of filters, however unlike the former it has geometrically spaced center frequencies that correspond to musical notes.

Other metrics were evaluated as well however only the RMSE-CQT was correlated with the quality of the generated audio. This (subjective) observation was initially made by listening to the audio samples and by comparing the

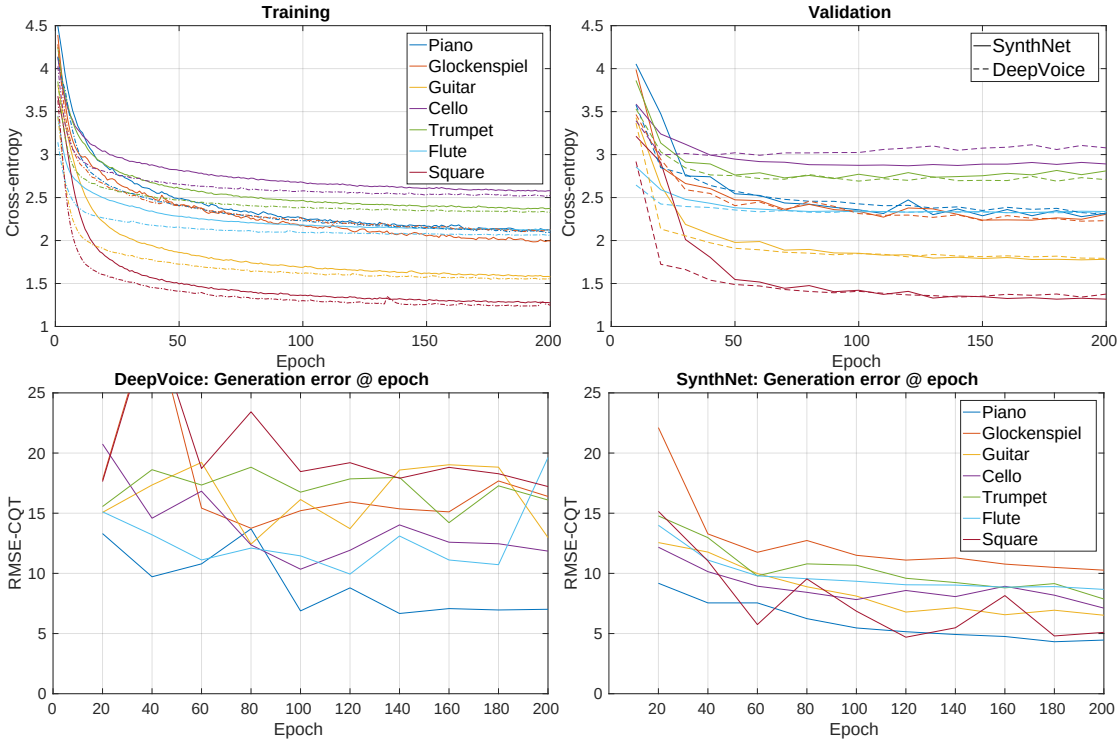


Figure 5.7: Seven networks are trained, each with a different harmonic style. Top, losses: training (left) validation (right). Bottom, RMSE-CQT: DeepVoice (left [Tbl. 5.3, col. 6]) and SynthNet (right [Tbl. 5.3, col. 8]). DeepVoice overfits for Glockenspiel (top right, dotted line). Convergence rate is measured via the RMSE-CQT, not the losses. The capacity of DeepVoice is larger, so the losses are steeper.

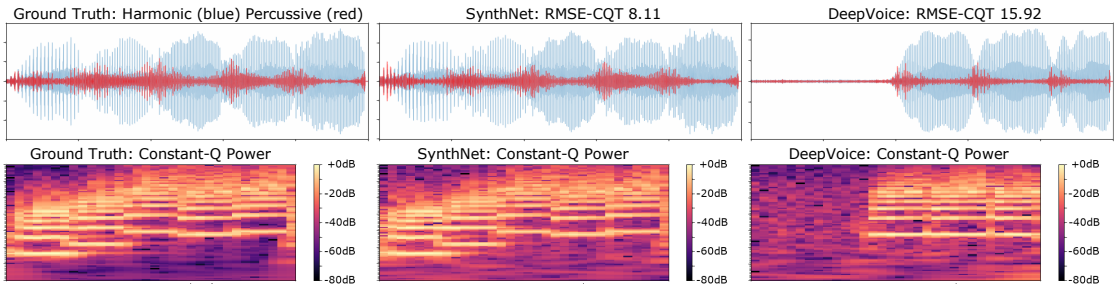


Figure 5.8: Left: 1 second of ground truth audio of Bach's BWV1007 Prelude, played with FluidSynth preset 56 Trumpet. Center: SynthNet high quality generated. Right: DeepVoice low quality generated showing delay. Further comparisons over other instrument presets are provided in Figure 5.9. I encourage the readers to listen to the samples here: `http://bit.ly/synthnet_appendix_a`

plots of the audio waveforms (Figure 5.8). Roughly speaking, as I also show in Figure 5.8 (top captions) and Figure 5.7 (lower plots), I find that a RMSE-CQT value below 10 corresponds to a generated sample of reasonable quality. The RMSE-CQT also penalizes temporal delays (Figure 5.8 - right) and is also correlated with the MOS (Table 5.3 and Table 5.5).

I generate every 20 epochs during training and compute the RMSE-CQT to check generation quality. Indeed, Figure 5.7 shows that the generated signals match the target audio better as the training progresses, while the losses flatten. However, occasionally the generated signals are shifted or the melody is slightly inaccurate - the wrong note is played (Figure 5.8 - right). This is not necessarily only a function of the network weight state since the generation process is stochastic. I set a fixed random seed at generation time, thus I only observe changes in the generated signal due to weight changes. To quantify error for one model, the RMSE-CQT is averaged over all epochs.

5.4.4 Hyperparameter selection

There are many possible configurations when it comes to the filter width F, the number of blocks B, and the maximum dilation rate R. The dilation rates per each block are: $\{2^0, 2^1, \ldots, 2^{R-1}\}$. In addition there is the choice of the number of residual and skip channels. For speech [9] use 64 residual channels and 256 skip channels, [50] use 512 residual channels and 256 skip channels, while [126] use 512 for both. These methods have receptive field $\Delta < 1$. Since the latter two works are also focused on music, I use 512 channels for both the residual and skip convolutions and set the final two convolutions to 512 and 384 channels respectively.

I hypothesize that it is better to maximize the receptive field Δ while minimizing the number of layers. Therefore, in the first experiments (Table 5.3) I limit the receptive field to 1 second and vary the other parameters according to Table 5.2. I have observed that the networks train faster and the quality is better when the length of the audio slice is maximized within GPU memory constraints.

Table 5.2: Three setups for filter, dilation and number of blocks resulting in a similar receptive field.

	Filter width F	**Num blocks** B	**Max dilation** R	**Receptive field** Δ
L24	3	2	12	1.0239 sec
L26	2	2	13	1.0240 sec
L48	2	4	12	1.0238 sec

Table 5.3: Mean RMSE-CQT and 95% confidence intervals (CIs). Two baselines are benchmarked for three sets of model hyperparameter settings (Table 5.2), all other parameters identical. One second of audio is generated every 20 epochs (over 200 epochs) and the error versus the target audio is measured and averaged over the epochs, per instrument. Total number of parameters and training time are also given. All waveforms and plots available here: `http://bit.ly/synthnet_table3`

	WaveNet			DeepVoice			SynthNet		
	L24	L26	L48	L24	L26	L48	L24	L26	L48
S01	16.56±4.67	14.83±5.39	18.15±2.88	10.80±1.66	9.32±1.97	17.28±2.19	6.30±1.01	5.96±1.10	*5.51±0.85*
S09	24.01±5.38	22.20±4.56	25.47±3.96	22.65±5.25	17.54±3.86	27.48±2.55	11.58±1.50	12.53±2.37	*10.91±1.40*
S24	17.68±3.05	18.95±4.13	19.30±1.26	18.03±1.58	16.33±1.79	19.19±1.25	8.00±1.71	8.53±1.51	*7.82±1.32*
S42	15.83±3.91	17.20±3.53	16.29±3.38	11.92±0.94	13.77±2.06	13.89±1.73	8.61±1.12	8.84±0.96	*8.33±0.74*
S56	18.50±2.23	17.25±2.98	22.89±1.73	17.04±0.34	17.16±1.05	21.45±1.37	*8.90±0.95*	10.37±1.41	8.97±1.42
S75	20.89±6.90	20.03±6.73	19.78±5.15	11.93±1.30	12.75±1.93	11.30±0.64	*9.68±1.22*	9.83±1.10	10.20±1.60
S80	27.73±2.29	26.74±3.92	26.96±4.71	20.41±1.80	20.09±2.91	20.95±2.77	*5.14±1.46*	7.66±2.31	7.91±2.41
All	20.02±1.79	**19.60±1.73**	21.35±1.48	16.18±1.30	**15.31±1.10**	18.57±1.36	**8.32±0.63**	9.10±0.70	8.52±0.62
Params	8.23e+7	6.18e+7	1.14e+8	8.90e+7	6.89e+7	1.27e+8	7.35e+6	7.80e+6	1.36e+7
Time	4d2h	4d2h	8+ days	4d10h	3d9h	8+ days	16 hours	16 hours	1d5h

It can be seen in Table 5.3 that SynthNet outperforms both baselines. Some instruments are more difficult to learn than others (also see Figure 5.7). This is also observable from listening to and visualizing the generated data (available here `http://bit.ly/synthnet_table3`).

The lowest errors for the first four instruments are observed for SynthNet L48 while the last three are lowest for SynthNet L24 (Table 5.3 slanted). This could be due to either an increased granularity over the frequency spectrum, provided by the extra layers of the L48 model or a better overlap. The best overall configuration is SynthNet L24. For DeepVoice and WaveNet, both L24 and L48 have more parameters (Table 5.3, second last row) and are slower to train, even though all setups have the same number of hidden channels (512) over both baseline architectures. This is because of the skip connections and associated convolutions.

Global conditioning I benchmark only DeepVoice L26 against SynthNet L24, with the difference that one model is trained to learn all 7 harmonic styles simultaneously (Table 5.4). This slows down training considerably. The errors are higher as opposed to learning one model per instrument, however SynthNet has the lowest error. I believe that increasing the number of residual channels would have resulted in lower error for both algorithms. I plan to explore this in future work.

Table 5.4: RMSE-CQT Mean and 95% CIs. All networks learn 7 harmonic styles simultaneously.

Experiment	Piano	Glockenspiel	Guitar	Cello	Trumpet	Flute	Square	All	Time
DeepVoice L26	14.01±1.41	19.68±3.29	16.10±1.60	13.80±2.11	18.68±2.04	15.40±3.22	15.64±1.76	16.19±0.91	12d3h
SynthNet L26	9.37±0.71	15.12±3.39	11.88±0.95	11.66±2.35	13.98±1.70	12.01±1.36	10.90±1.17	**12.13±0.74**	5d23h

5.4.5 MOS listening tests

Given the results from Table 5.3, I benchmark the best performing setups: WaveNet L26, DeepVoice L26 and SynthNet L24. For these experiments, I generate samples from multiple songs using the converged models from all instruments. I generate 5 seconds of audio from Bach's Cello suites not seen during training,

namely Part 1 of Suite No. 1 in G major (BWV 1007), Part 1 of Suite No. 2 in D minor (BWV 1008) and Part 1 of Suite No. 3 in C major (BWV 1009) which cover a broad range of notes and rhythm variations.

Table 5.5 shows that the samples generated by SynthNet are rated to be almost twice as better than the baselines, over all harmonic styles. By listening to the samples (`http://bit.ly/synthnet_mostest`), one can observe that Piano is the best overall learned model, while the baseline algorithms have trouble playing the correct melody over longer time spans for other styles. I would also like to remind the reader that all networks have been trained with only 9 minutes of data.

Table 5.5: Listening MOS and 95% CIs. 5 seconds of audio are generated from 3 musical pieces (Bach's BWV 1007, 1008 and 1009), over 7 instruments for the best found models. Subjects are asked to listen to the ground truth reference, then rate samples from all 3 algorithms simultaneously. 20 ratings are collected for each file. Audio and plots here: `http://bit.ly/synthnet_mostest`

Experiment	Piano	Glockenspiel	Guitar	Cello	Trumpet	Flute	Square	All
WaveNet L26	2.22±0.25	2.48±0.23	2.18±0.25	2.37±0.28	2.18±0.29	2.37±0.22	2.30±0.09	2.30±0.10
DeepVoice L26	2.55±0.32	1.85±0.23	2.30±0.39	2.62±0.27	2.28±0.32	2.20±0.25	1.87±0.03	2.24±0.11
SynthNet L24	4.75±0.14	4.45±0.17	4.30±0.19	4.50±0.15	4.25±0.18	4.15±0.21	4.10±0.16	**4.36±0.07**

5.5 Discussion

In the current chapter I gave some insights into the learned representations of generative convolutional models. I tested the hypothesis that the first causal layer learns fundamental frequencies. I validated this empirically, arriving at the SynthNet architecture which converges faster and produces higher quality audio.

The method is able to simultaneously learn the characteristic harmonics of a musical instrument (timbre) and a joint embedding between notes and the corresponding fundamental frequencies. While I focus on music, I believe that SynthNet can also be successfully used for other time series problems. I plan to investigate this in future work.

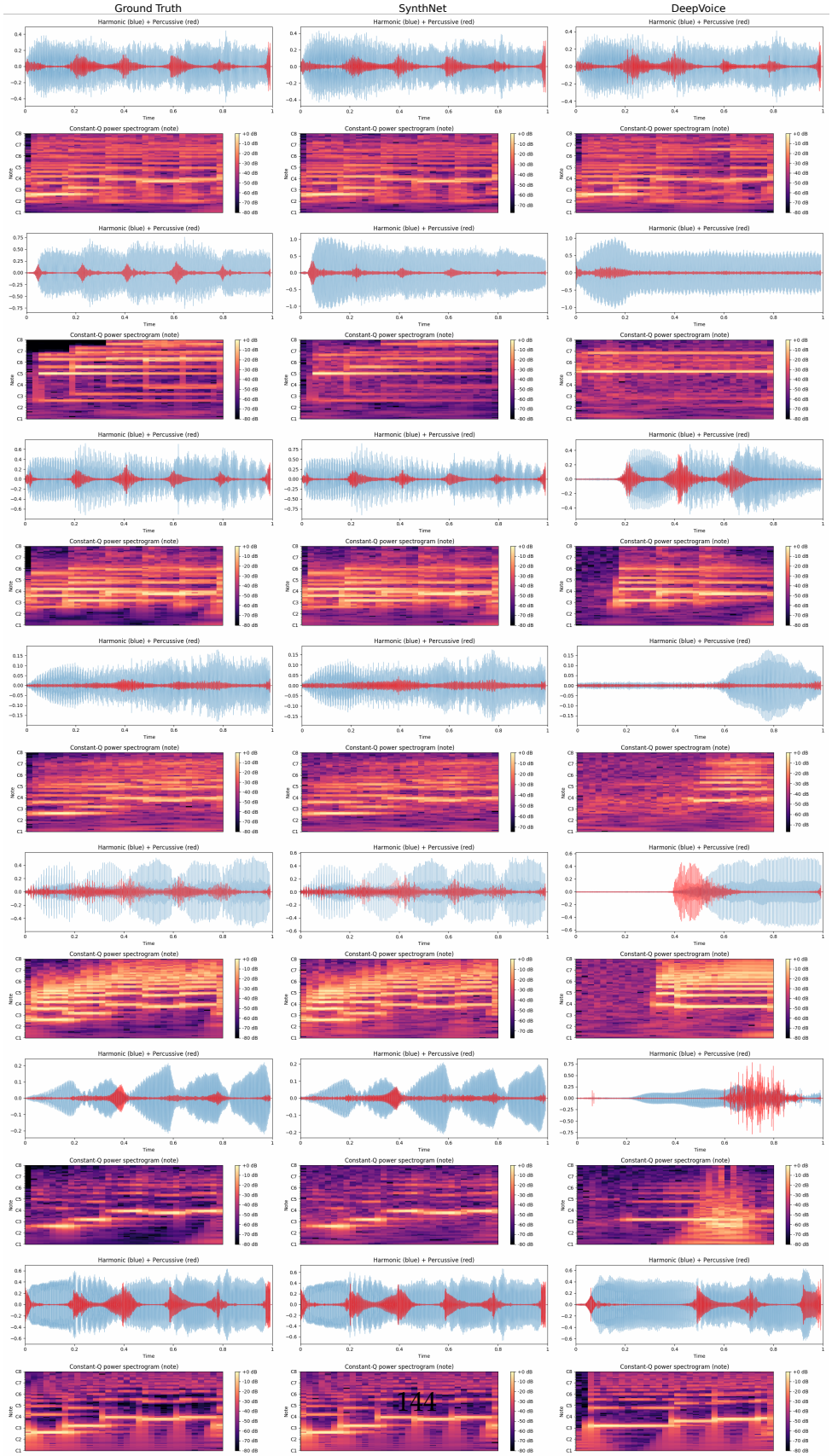


Figure 5.9: Audio samples and visualizations here: `http://bit.ly/synthnet_appendix_a`

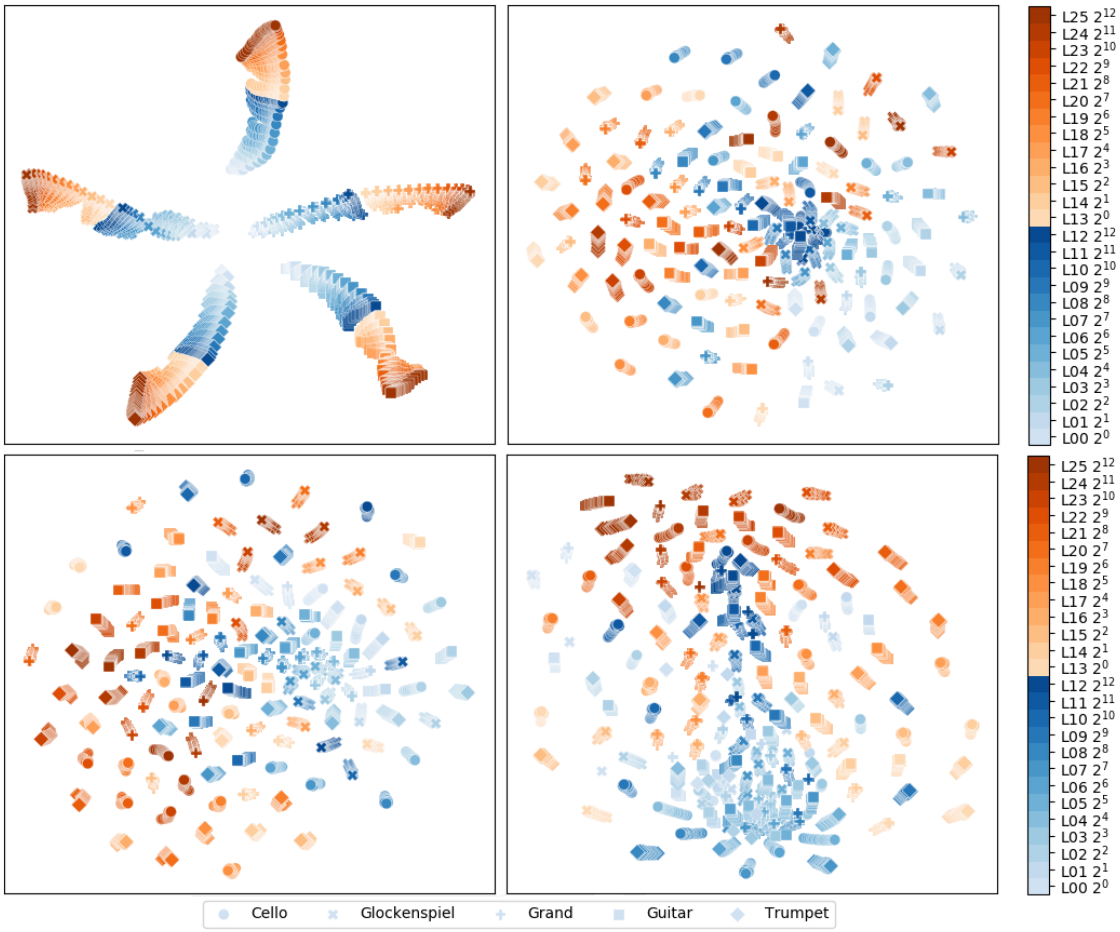


Figure 5.10: Gram matrices extracted during training, every 20 epochs. Top left: extracted from Equation 5.3. Top right: extracted from Equation 5.2. Bottom left: extracted from the filter part of Equation 5.2. Bottom right: extracted from the gate part of Equation 5.2.

Chapter 6

Conclusions

This chapter summarises the contributions of this thesis and subsequently suggests possible further research avenues.

6.1 Summary of contributions

The rise of IoT fuels the need for robust, versatile and efficient algorithms for time series applications. This thesis is focused on causal CNNs for prediction (modelled as both classification and regression) for MTL in ST problems and for generative autoregressive models.

In Chapter 3 causal CNNs are benchmarked against other methods for MTL as a peak traffic forecasting problem (imbalanced classification) and shows that it is beneficial to leverage old data and also information from neighbouring sensors. The experiments show that increasing the receptive field (past temporal context) is beneficial and the performance gains reach a plateau where the performance gains are not justified by the increased computational load. Furthermore, discarding old data does not increase accuracy which suggests that the volume and variance of data is important. This chapter also shows that in some cases where specific temporal sub problems (i.e. weekdays only) are sought, the accuracy can be further increased. One major inconvenience with such problems is picking the threshold for peak traffic classification. This parameter should be adapted dynamically according to the task and the time of

the day. The thresholding procedure introduces more problems than it solves, which is why the next chapter models the problem as regression.

Chapter 4 introduces the TRU-VAR framework that applies causal CNNs to MTL problems in the context of spatio-temporal data, for continuous valued outputs. In the context of IoT, deployability, performance and flexibility, this chapter also proposes some properties for large scale network wide traffic forecasting. It also provides a broad review of the literature. Data quality, ingress and preprocessing is also discussed and alternatives for defining the task relationship matrix (TDAM). TRU-VAR is proposed and the overall error for the entire 1000+ tasks is compared against pure univariate methods over two quantitatively and qualitatively different datasets. TRU-VAR with Causal CNNs outperform all other state of the art methods and the baselines. Furthermore, when the prediction horizon is moved aheadn further in time (i.e. problem is more difficult) the TRU-VAR causal CNN had the lowest error, even when the forecast horizon is increased up to two hours, for both datasets. This shows that causal CNNs are effective in multi-task learning applications with spatio-temporal data. The method can be trained online, with efficient CNNs, has a low complexity, is non-static and robust towards changes (data sources or structure) thus scales well, is efficient and easy to redeploy. Given that the error increases linearly within reasonable bounds (Figure 4.12) for up to two hours, the same method can be applied to other problems, such as natural disaster prevention (e.g. river flood forecasting), telecommunications (e.g. antenna load forecasts), networking (e.g. forecasts for routing), finance to name a few. In general, the method can be applied to any type of dataset consisting of multivariate timeseries, where the constraints are known a priori or can be inferred from the data to construct the topology matrix as shown in section 2.2.

Chapter 5 introduces a new generative model - SynthNet - which is more accurate and faster than the baselines, as demonstrated for 7 instrument timbres. This is demonstrated with subjective listening tests as well as a quantitative measurment, the RMSE-CQT. Furthermore, this is the first time that a synthesizer (or vocoder for voice) is learned directly from aligned music, based on aligned piano roll data with raw waveforms. Previous work required the carefully annotated individual notes for each note and velocity, which is unpractical

for real instruments. This chapter also provides a hypothesis for what autoregressive generative CNN models learn and gives some insights into the learned representations of generative convolutional models. The method is able to simultaneously learn the characteristic harmonics of a musical instrument (timbre) and a joint embedding between notes and the corresponding fundamental frequencies. While this work is focused on music, the applications of this algorithm is not limited to audio or music.

6.2 Thesis limitations and future work

A few limitations strictly applied to the MTL problems is that here data from only one domain (traffic sensors) is used during training and forecasting. In actual deployment of these models, data from multiple sources such as moving sensors (people's phones), weather information, calendar events and so on can be used in order to improve the prediction accuracy.

In addition, the topological adjacency matrix (TDAM) is based on only the first order degree road connections. Evidently, I could have also opted to select higher order degrees e.g. $G^2(i)$, for designing the TDAM. It would be interesting to observe the effects in urban settings since the number of such neighbours can increase exponentially as the degree increases. However, the matrix could be heuristically adapted according to the number of first order connections, in the case of highways where the degree of a node can be low. This rule of not having the same degree for each node has not been explored in the current work.

Another extension of this work is learning the task relatedness matrix for regularization in the MTL context. While a good starting candidate for such a regularizer is given in section 2.2, this is not tested on either of the datasets. It would be interesting to observe the effects of such regularization strategies for both one large MTL task as well as several smaller MTL tasks. The VicRoads dataset is an ideal candidate to study this problem since the topology of the roads is already known. Then, since the ground truth is known the question arises whether the structure of the task relatedness can be properly learned from the data and given the ground truth, this can also be verified.

Another avenue not explored in this thesis is knowledge distillation for MTL

problems and the effect of regularization in this context. Knowledge distillation is an indirect approach where a trained large model or an ensemble is used to supervise the training of a smaller model. It is usually the case that the smaller models achieve comparative performance to the original models. This is especially useful for embedded low power computing.

SynthNet can be applied to forecasting as well and is most suitable for high granularity data (even milliseconds if available), although this is not demonstrated in this thesis. SynthNet would also be a perfect candidate for multiple domain data, since it naturally allows the prediction to be conditioned on additional information.

Although not demonstrated, SynthNet can be used to generate artificial / fake / adversarial data. While the utility of this is not obvious for traffic data, it may be more evident for computer networks. There, artificial data can be created to defend by spoofing against Denial of Service (DoS) attacks. The latter need to be timed at high network loads and at specific nodes in order to bring networks down. Even fake nodes could be generated as part of the network, in order to attract the attention of attackers. In the anti-virus world this concept is called HoneyPot - where some systems that are not used in production are deliberately weakened to attract malicious software. This allows the defenders to monitor the network and prepare for any attacks.

Bibliography

[1] Baher Abdulhai, Himanshu Porwal, and Will Recker. Short-term traffic flow prediction using neuro-genetic algorithms. *ITS Journal-Intelligent Transportation Systems Journal*, 7(1):3–41, 2002.

[2] Jin Young Ahn, Eunjeong Ko, and EunYi Kim. Predicting spatiotemporal traffic flow based on support vector regression and bayesian classifier. In *Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on*, pages 125–130. IEEE, 2015.

[3] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[4] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

[5] Tarique Anwar et al. Spatial partitioning of large urban road networks. In *EDBT*, pages 343–354, 2014.

[6] Sarath Chandar AP, Stanislas Lauly, Hugo Larochelle, Mitesh Khapra, Balaraman Ravindran, Vikas C Raykar, and Amrita Saha. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*, pages 1853–1861, 2014.

[7] Van Arem et al. Recent advances and applications in the field of short-term traffic forecasting. *Int. Jrnl of Forecasting*, 13(1):1–12, 1997.

[8] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

[9] Sercan O Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Andrew Ng, Jonathan Raiman, et al. Deep voice: Real-time neural text-to-speech. *arXiv preprint arXiv:1702.07825*, 2017.

[10] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[11] Muhammad Tayyab Asif, Justin Dauwels, Chong Yang Goh, Ali Oran, Esmail Fathi, Muye Xu, Menoth Mohan Dhanya, Nikola Mitrovic, and Patrick Jaillet. Spatiotemporal patterns in large-scale traffic speed prediction. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):794–804, 2014.

[12] Dimitrios Asteriou et al. *Applied econometrics*, pages 265–286. Palgrave Macmillan, 2011.

[13] George Athanasopoulos, DS Poskitt, and Farshid Vahid. Two canonical varma forms: Scalar component models vis-à-vis the echelon form. *Econometric Reviews*, 31(1):60–83, 2012.

[14] Tsz-Chiu Au, Shun Zhang, and Peter Stone. Autonomous intersection management for semi-autonomous vehicles. *Handbook of Transportation. Routledge, Taylor & Francis Group*, 2015.

[15] E Michael Azoff. *Neural network time series forecasting of financial markets*. John Wiley & Sons, Inc., 1994.

[16] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[17] Luca Baldassarre, Lorenzo Rosasco, Annalisa Barla, and Alessandro Verri. Multi-output learning via spectral filtering. *Machine learning*, 87(3):259–301, 2012.

[18] VICTOR Blue et al. Neural network freeway travel time estimation. In *Intell. Eng. Syst. Artificial Neural Nets (Saint Louis, Mo.) Proc. of*, volume 4, 1994.

[19] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[20] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[21] Judith C Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[22] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[23] Enrique Castillo, José María Menéndez, and Santos Sánchez-Cambronero. Predicting traffic flow using bayesian networks. *Transportation Research Part B: Methodological*, 42(5):482–509, 2008.

[24] Manoel Castro-Neto, Young-Seon Jeong, Myong-Kee Jeong, and Lee D Han. Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert systems with applications*, 36(3):6164–6173, 2009.

[25] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Linear algorithms for online multitask classification. *Journal of Machine Learning Research*, 11(Oct):2901–2934, 2010.

[26] B Gültekin Çetiner, Murat Sari, and Oğuz Borat. A neural network based traffic-flow prediction model. *Mathematical and Computational Applications*, 15(2):269–278, 2010.

[27] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[28] Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1189–1198. ACM, 2010.

[29] Chenyi Chen, Jianming Hu, Qiang Meng, and Yi Zhang. Short-time traffic flow prediction with arima-garch model. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 607–612. IEEE, 2011.

[30] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[31] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

[32] Tao Cheng et.al. Spatio-temporal autocorrelation of road network data. *Jrnl of Geographical Syst.*, 14(4):389–413, 2012.

[33] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[34] Roland Chrobok et al. Three categories of traffic data: Historical, current, and predictive. In *Control in Transp. Syst., Proc. of the 9th IFAC Symp.*, pages 250–255, 2000.

[35] Stepen Clark et al. The use of neural networks and time series models for short term traffic forecasting: a comparative study. In *European Transp., Highways And Planning 21st Annual Meeting*, volume P363, 1993.

[36] Stephen Clark. Traffic prediction using multivariate nonparametric regression. *Jrnl of Transp. Eng.*, 129(2):161–168, 2003.

[37] Serdar Çolak, Antonio Lima, and Marta C González. Understanding congested travel in urban areas. *Nature communications*, 7, 2016.

[38] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[39] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[40] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE, 2013.

[41] Hussein Dia. An object-oriented neural network approach to short-term traffic forecasting. *European Jrnl of Oprn Res.*, 131(2):253–261, 2001.

[42] Francis X Diebold and Robert S Mariano. Comparing predictive accuracy. *Journal of Business & economic statistics*, 2012.

[43] Sander Dieleman, Aäron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. *arXiv preprint arXiv:1806.10474*, 2018.

[44] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[45] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[46] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[47] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[48] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018.

[49] Mark S Dougherty et al. Short-term inter-urban traffic forecasts using neural networks. *Int. Jrnl of forecasting*, 13(1):21–31, 1997.

[50] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint arXiv:1704.01279*, 2017.

[51] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

[52] Julian Faraway and Chris Chatfield. Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(2):231–250, 1998.

[53] F Dan Foresee and Martin T Hagan. Gauss-newton approximation to bayesian learning. In *Neural Networks, 1997., International Conference on*, volume 3, pages 1930–1935. IEEE, 1997.

[54] Gaetano Fusco, Chiara Colombaroni, Luciano Comelli, and Natalia Isaenko. Short-term traffic predictions on large urban traffic networks: applications of network-based machine learning models and dynamic traffic assignment models. In *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on*, pages 93–101. IEEE, 2015.

[55] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[56] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[57] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[58] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[59] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[60] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[61] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[62] R Guidotti, M Nanni, S Rinzivillo, D Pedreschi, and F Giannotti. Never drive alone: Boosting carpooling with network analysis. *Information Systems*, 2016.

[63] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[64] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[65] Coşkun Hamzaçebi. Improving artificial neural networks' performance in seasonal time series forecasting. *Information Sciences*, 178(23):4550–4559, 2008.

[66] Johanna Hansen, Kyle Kastner, Aaron Courville, and Gregory Dudek. Planning in dynamic environments with conditional autoregressive models. 2018.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[68] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[69] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[70] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[71] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[72] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[73] Per Högberg. Estimation of parameters in models for traffic prediction: a non-linear regression approach. *Transportation Research*, 10(4):263–265, 1976.

[74] Haikun Hong, Wenhao Huang, Xingxing Xing, Xiabing Zhou, Hongyu Lu, Kaigui Bian, and Kunqing Xie. Hybrid multi-metric k-nearest neighbor regression for traffic flow prediction. In *2015 IEEE 18th International*

Conference on Intelligent Transportation Systems, pages 2262–2267. IEEE, 2015.

[75] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[76] Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S Jensen. Stochastic weight completion for road networks using graph convolutional networks. In *Proceedings of 34th Ieee International Conference on Data Engineering, Icde 2019*. IEEE, 2018.

[77] Jilin Hu, Chenjuan Guo, Bin Yang, Christian S Jensen, and Lu Chen. Recurrent multi-graph neural networks for travel cost prediction. *arXiv preprint arXiv:1811.05157*, 2018.

[78] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[79] Rob J Hyndman, Yeasmin Khandakar, et al. Automatic time series for forecasting: the forecast package for r. Technical report, Monash University, Department of Econometrics and Business Statistics, 2007.

[80] T Jensen and SK Nielsen. Calibrating a gravity model and estimating its parameters using traffic volume counts. In *5th Conference of Universities' Transport Study Groups, University College, London*, 1973.

[81] Pushkin Kachroo and Shankar Sastry. Traffic assignment using a density-based travel-time function for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 17(5):1438–1447, 2016.

[82] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[83] Yiannis Kamarianakis and Poulicos Prastacos. Space–time modeling of traffic flow. *Computers & Geosciences*, 31(2):119–133, 2005.

[84] Yiannis Kamarianakis et al. Forecasting traffic flow conditions in an urban network: comparison of multivariate and univariate approaches. *Transp. Res. Record*, (1857):74–84, 2003.

[85] Yiannis Kamarianakis et al. Real-time road traffic forecasting using regime-switching space-time models and adaptive lasso. *Applied Stochastic Models in Business and Industry*, 28(4):297–315, 2012.

[86] Matthew G Karlaftis and Eleni I Vlahogianni. Statistical methods versus neural networks in transportation research: Differences, similarities and some insights. *Transportation Research Part C: Emerging Technologies*, 19(3): 387–399, 2011.

[87] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 3, 2017.

[88] J Kihoro, R Otieno, and C Wafula. Seasonal time series forecasting: A comparative study of arima and ann models. *AJST*, 5(2), 2004.

[89] Seyoung Kim and Eric P Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *ICML*, pages 543–550, 2010.

[90] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[91] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[92] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[93] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse

autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[94] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

[95] Eunjeong Ko, Jinyoung Ahn, and Eun Yi Kim. 3d markov process for traffic flow prediction in real-time. *Sensors*, 16(2):147, 2016.

[96] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[98] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

[99] S Vasantha Kumar and Lelitha Vanajakshi. Short-term traffic flow prediction using seasonal arima model with limited input data. *European Transport Research Review*, 7(3):1–9, 2015.

[100] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

[101] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[102] Eun-Mi Lee, Jai-Hoon Kim, and Won-Sik Yoon. Traffic speed prediction under weekday, time, and neighboring links' speed: Back propagation neural network approach. In *International Conference on Intelligent Computing*, pages 626–635. Springer, 2007.

[103] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

[104] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[105] Moshe Levin and Yen-Der Tsao. On forecasting freeway occupancies and volumes (abridgment). *Transportation Research Record*, (773), 1980.

[106] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Graph convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[107] Thomas Liebig, Nico Piatkowski, Christian Bockermann, and Katharina Morik. Dynamic route planning with real-time traffic predictions. *Information Systems*, 2016.

[108] Marco Lippi, Marco Bertini, and Paolo Frasconi. Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning. *Intelligent Transportation Systems, IEEE Transactions on*, 14(2):871–882, 2013.

[109] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[110] Dana E Low. A new approach to transportation systems modeling. *Traffic quarterly*, 26(3), 1972.

[111] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech*, pages 436–440, 2013.

[112] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogério Schmidt Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, volume 1, page 6, 2017.

[113] Helmut Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[114] Yisheng Lv, Shuming Tang, and Hongxia Zhao. Real-time highway traffic accident prediction based on the k-nearest neighbor method. In *2009 International Conference on Measuring Technology and Mechatronics Automation*, volume 3, pages 547–550. IEEE, 2009.

[115] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: A deep learning approach. *Intelligent Transportation Systems, IEEE Transactions on*, 16(2):865–873, 2015.

[116] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[117] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[118] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[119] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[120] Wanli Min et al. Real-time road traffic prediction with spatio-temporal correlations. *Transp. Res. Part C: Emerging Tech.*, 19(4):606–616, 2011.

[121] Xinyu Min, Jianming Hu, Qi Chen, Tongshuai Zhang, and Yi Zhang. Short-term traffic flow forecasting of urban network based on dynamic starima model. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6. IEEE, 2009.

[122] Xinyu Min et al. Urban traffic network modeling and short-term traffic flow forecasting based on gstarima model. In *Intell. Transp. Syst. (ITSC), 13th Int. Conf. on*, pages 1535–1540. IEEE, 2010.

[123] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016.

[124] Nikola Mitrovic, Muhammad Tayyab Asif, Justin Dauwels, and Patrick Jaillet. Low-dimensional models for compressed sensing and prediction of large-scale traffic data. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2949–2954, 2015.

[125] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[126] Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A universal music translation network. *arXiv preprint arXiv:1805.07848*, 2018.

[127] Luis Moreira-Matias and Francesco Alesiani. Drift3flow: Freeway-incident prediction using real-time learning. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 566–571. IEEE, 2015.

[128] Luís Moreira-Matias, João Gama, Michel Ferreira, João Mendes-Moreira, and Luis Damas. Time-evolving od matrix estimation using high-speed gps data streams. *Expert Systems With Applications*, 44:275–288, 2016.

[129] Kevin P Murphy. *Machine learning: a probabilistic perspective*. 2012.

[130] Yurii Nesterov et al. Gradient methods for minimizing composite objective function, 2007.

[131] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[132] Simon Oh, Young-Ji Byon, Kitae Jang, and Hwasoo Yeo. Short-term travel-time prediction on highway: a review of the data-driven approach. *Transport Reviews*, 35(1):4–32, 2015.

[133] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[134] Zhijian Ou. A review of learning with deep generative models from perspective of graphical modeling. *arXiv preprint arXiv:1808.01630*, 2018.

[135] Jungme Park et al. Real time vehicle speed prediction using a neural network traffic model. In *Neural Networks (IJCNN), Int. Joint Conf. on*, pages 2991–2996. IEEE, 2011.

[136] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, number EPFL-CONF-199822, 2014.

[137] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[138] Marek Rei. Semi-supervised multitask learning for sequence labeling. *arXiv preprint arXiv:1704.07156*, 2017.

[139] William Remus and Marcus O'Connor. Neural networks for time-series forecasting. In *Principles of forecasting*, pages 245–256. Springer, 2001.

[140] John Rice and Erik Van Zwet. A simple and effective method for predicting travel times on freeways. *IEEE Transactions on Intelligent Transportation Systems*, 5(3):200–207, 2004.

[141] Bernardino Romera-Paredes, Andreas Argyriou, Nadia Berthouze, and Massimiliano Pontil. Exploiting unrelated tasks in multi-task learning. In *International Conference on Artificial Intelligence and Statistics*, pages 951–959, 2012.

[142] Avishek Saha, Piyush Rai, Hal DaumÃ, Suresh Venkatasubramanian, et al. Online learning of multiple tasks and their relationships. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 643–651, 2011.

[143] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[144] Athanasios Salamanis, Dionysios D Kehagias, Christos K Filelis-Papadopoulos, Dimitrios Tzovaras, and George A Gravvanis. Managing spatial graph dependencies in large volumes of traffic data for travel-time prediction. *IEEE Transactions on Intelligent Transportation Systems*, 17(6): 1678–1687, 2016.

[145] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[146] Florin Schimbinschi, Lambert Schomaker, and Marco Wiering. Ensemble methods for robust 3d face recognition using commodity depth sensors. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 180–187. IEEE, 2015.

[147] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[148] Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. In *Artificial Intelligence and Statistics*, pages 436–443, 2007.

[149] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*, 2017.

[150] William A Sethares. *Tuning, timbre, spectrum, scale*. Springer Science & Business Media, 2005.

[151] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[152] Brian Smith et al. Short-term traffic flow prediction: neural network approach. *Transp. Res. Record*, (1453), 1994.

[153] Brian L Smith and Michael J Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of transportation engineering*, 123(4): 261–266, 1997.

[154] Brian L Smith, Billy M Williams, and R Keith Oswald. Comparison of parametric and nonparametric models for traffic flow forecasting. *Transportation Research Part C: Emerging Technologies*, 10(4):303–321, 2002.

[155] Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[156] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[157] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[158] Anthony Stathopoulos and Matthew G Karlaftis. A multivariate state space approach for urban traffic flow modeling and prediction. *Transportation Research Part C: Emerging Technologies*, 11(2):121–135, 2003.

[159] Antony Stathopoulos, Loukas Dimitriou, and Theodore Tsekeris. Fuzzy modeling approach for combined forecasting of urban traffic flow. *Computer-Aided Civil and Infrastructure Engineering*, 23(7):521–535, 2008.

[160] Fei Su, Honghui Dong, Limin Jia, Yong Qin, and Zhao Tian. Long-term forecasting oriented to urban expressway traffic situation. *Advances in Mechanical Engineering*, 8(1):1687814016628397, 2016.

[161] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pages 1601–1608, 2009.

[162] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[163] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[164] WY Szeto, Bidisha Ghosh, Biswajit Basu, and Margaret O'Mahony. Multivariate traffic forecasting technique using cell transmission model and sarima model. *Journal of Transportation Engineering*, 135(9):658–667, 2009.

[165] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[166] Zaiyong Tang, Chrys De Almeida, and Paul A Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *Simulation*, 57(5):303–310, 1991.

[167] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. *arXiv preprint arXiv:1612.07695*, 2016.

[168] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[169] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[170] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[171] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

[172] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[173] JWC Van Lint and CPIJ Van Hinsbergen. Short-term traffic and travel time prediction models. *Artificial Intelligence Applications to Critical Transportation Issues*, 22:22–41, 2012.

[174] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[175] Pravin Pratap Varaiya. *Freeway performance measurement system: Final report*. Citeseer, 2001.

[176] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[177] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[178] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[179] Eleni I Vlahogianni, John C Golias, and Matthew G Karlaftis. Short-term traffic forecasting: Overview of objectives and methods. *Transport reviews*, 24(5):533–557, 2004.

[180] Eleni I Vlahogianni, Matthew G Karlaftis, and John C Golias. Optimized and meta-optimized neural networks for short-term traffic flow prediction: a genetic approach. *Transportation Research Part C: Emerging Technologies*, 13(3):211–234, 2005.

[181] Eleni I Vlahogianni, Matthew G Karlaftis, and John C Golias. Short-term traffic forecasting: Where we are and where we're going. *Transportation Research Part C: Emerging Technologies*, 43:3–19, 2014.

[182] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[183] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[184] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013.

[185] Yibing Wang et al. Real-time freeway traffic state estimation based on extended kalman filter: Adaptive capabilities and real data testing. *Transp. Res. Part A: Policy and Practice*, 42(10):1340–1358, 2008.

[186] Marco A Wiering and Lambert RB Schomaker. Multi-layer support vector machines. *Regularization, optimization, kernels, and support vector machines*, page 457, 2014.

[187] Billy Williams. Multivariate vehicular traffic flow prediction: evaluation of arimax modeling. *Transportation Research Record: Journal of the Transportation Research Board*, (1776):194–200, 2001.

[188] Stephen J Wright, Robert D Nowak, and Mário AT Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

[189] Chun-Hsin Wu, Jan-Ming Ho, and Der-Tsai Lee. Travel-time prediction with support vector regression. *IEEE transactions on intelligent transportation systems*, 5(4):276–281, 2004.

[190] Yao-Jan Wu, Feng Chen, Chang-Tien Lu, and Shu Yang. Urban traffic flow prediction using a spatio-temporal random effects model. *Journal of Intelligent Transportation Systems*, 20(3):282–293, 2016.

[191] Yuanchang Xie et al. Short-term traffic volume forecasting using kalman filter with discrete wavelet decomposition. *Computer-Aided Civil and Infrastructure Eng.*, 22(5):326–334, 2007.

[192] Yanyan Xu, Hui Chen, Qing-Jie Kong, Xi Zhai, and Yuncai Liu. Urban traffic flow prediction: a spatio-temporal variable selection-based approach. *Journal of Advanced Transportation*, 2015.

[193] Bin Yang, Chenjuan Guo, and Christian S Jensen. Travel cost inference from sparse, spatio temporally correlated time series using markov models. *Proceedings of the VLDB Endowment*, 6(9):769–780, 2013.

[194] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Ijcai*, volume 15, pages 3995–4001, 2015.

[195] Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391*, 2016.

[196] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[197] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[198] Dehuai Zeng, Jianmin Xu, Jianwei Gu, Liyan Liu, and Gang Xu. Short term traffic flow prediction based on online learning svr. In *Power Electronics and Intelligent Transportation System, 2008. PEITS'08. Workshop on*, pages 616–620. IEEE, 2008.

[199] Cun-Hui Zhang, Jian Huang, et al. The sparsity and bias of the lasso selection in high-dimensional linear regression. *The Annals of Statistics*, 36 (4):1567–1594, 2008.

[200] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[201] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[202] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

[203] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision*, pages 94–108. Springer, 2014.

[204] Weizhong Zheng, Der-Horng Lee, and Qixin Shi. Short-term freeway traffic flow prediction: Bayesian combined neural network approach. *Journal of transportation engineering*, 132(2):114–121, 2006.

[205] Hui Zou et al. Regularization and variable selection via the elastic net. *Jrnl of the Royal Stat. Soc.: Series B (Stat. Methodology)*, 67(2):301–320, 2005.